Hindawi
Wireless Communications and Mobile Computing
Volume 2023, Article ID 1357343, 12 pages
https://doi.org/10.1155/2023/1357343

Research Article

A Deep Learning-Based Algorithm for Energy and Performance Optimization of Computational Offloading in Mobile Edge Computing

Israr Khan,[1] **Salman Raza**,[2] **Waheed ur Rehman**,[1,3] **Razaullah Khan**,[1,4] **Kiran Nahida**,[5] **and Xiaofeng Tao**[1]

[1]*National Engineering Laboratory for Mobile Network Technologies, Beijing University of Posts and Telecommunications, Beijing 100876, China*
[2]*Department of Computer Science, National Textile University Faisalabad, Pakistan*
[3]*Department of Computer Science, University of Peshawar, Peshawar, Pakistan*
[4]*Department of Computer Science, University of Engineering and Technology Mardan, Pakistan*
[5]*Beijing Laboratory of Advanced Information Network and the Beijing Key Laboratory of Network System Architecture and Convergence, Beijing University of Posts and Telecommunications, China*

Correspondence should be addressed to Xiaofeng Tao; taoxf@bupt.edu.cn

Received 16 August 2022; Revised 28 February 2023; Accepted 6 April 2023; Published 4 May 2023

Academic Editor: Fuliang Li

Mobile edge computing (MEC) has produced incredible outcomes in the context of computationally intensive mobile applications by offloading computation to a neighboring server to limit the energy usage of user equipment (UE). However, choosing a pool of application components to offload in addition to the volume of data transfer along with the latency in communication is an intricate issue. In this article, we introduce a novel energy-efficient offloading scheme based on deep neural networks. The proposed scheme trains an intelligent decision-making model that picks a robust pool of application components. The selection is based on factors such as the remaining UE battery power, network conditions, the volume of data transfer, required energy by the application components, postponements in communication, and computational load. We have designed the cost function taking all the mentioned factors, get the cost for all conceivable combinations of component offloading decisions, pick the robust decisions over an extensive dataset, and train a deep neural network as a substitute for the exhaustive computations associated. Model outcomes illustrate that our proposed scheme is proficient in the context of accuracy, root mean square error (RMSE), mean absolute error (MAE), and energy usage of UE.

1. Introduction

Nowadays, technological advancement is emerging drastically fast, along with numerous applications. The future wireless communication grasps various application setups in the form of augmented reality and cognitive assistance. Billions of smart devices are equipped with high computational resources and considerable memory size but also necessitate performing larger number of tasks. To tackle these challenges, researchers suggested a substantial solution, namely, mobile cloud computing (MCC). Aside from providing a solution, it also imposes an extra load on backhaul and radio mobile networks which consequences into a high or variable latency to remote data centers. Therefore, the concept of mobile edge computing (MEC) emerged which offers users storage and computing resources. It also minimizes the load on network resources, the energy consumption of user equipment (UE), and the network delay. Within the MEC, resource allocation along with offloading approaches precisely afflicts the performance of the framework which has turned out to be a research trend recently.

In [1], a two-phase traffic distribution method was projected for mobile edge server (MES) computing resources and mutually optimizing channel bandwidth to minimalize

latency. In [2], an algorithm by the game theory was offered to mutually optimize channel bandwidth and MES computing resources to minimize energy consumption and overall time. Within MEC, enhancing the mobile applications' performance mainly relies on effective task offloading decisions [3–6]. Thus, offloading decision-making has shown promising results over the past few years [7–12]. The authors in [11] measured the capacity limitations of backhaul links and real-time and maximum delay limitations of users and presented an unloading strategy to minimize the overall network energy consumption. In [12], the authors presented a computation offloading strategy considering energy perception through the consumption of weighing energy and time delay. The authors also contributed the residual energy of smart device battery within the characterization of the weighted factor of delay and energy consumption, decreasing consumption of the entire system. Both approaches mentioned above neglected to allocate computing resources and a limited spectrum. In [13], the authors presented an adaptive resource allocation and task offloading system for MEC. The proposed algorithm utilized deep reinforcement learning (DRL) technique to identify whether or not a task demands to remain offloaded and allocate computing resources for the particular task. However, this technique has drawbacks such as tough to regulate parameters and long training time. In the scenario with multiple resources, [14] considered an algorithm based on task scheduling of energy consumption minimization particle swarm optimization aimed at multiple resources corresponding to decrease edge terminal equipment energy consumption. In [15], a privacy viewpoint computing offloading algorithm is proposed, which is based on the Lyapunov optimization theory. The authors in [16] considered deep learning task offloading to deploy deep learning applications along with improving network energy consumption, a group of sparse beamforming structure based upon mixed L1/L2 norms.

While there are various works of literature concerning the computation offloading system for a user single-cell MEC framework, the vast majority of earlier offloading works based on machine learning (ML) presume either an infinite quantity of accessible communication or computation resources in cloudlet or coarse-grained computation offloading [1, 2]. This motivates us to propose a new offloading method that utilizes deep neural networks to achieve energy efficiency. The proposed approach trains an intelligent decision-making model that selects a reliable set of application components based on various factors, including remaining battery power, network conditions, data transfer volume, energy requirements of the components, communication delays, and computational load. To accomplish this, a cost function is developed that considers all these factors, and then, the most robust offloading decisions are chosen from a large dataset. Finally, a deep neural network is trained to serve as a more efficient substitute for the exhaustive computations required by the cost function.

The main contributions of this article are the following:

(i) To deal with the offloading issue, an effective technique is proposed, which selects an optimum component's part to offload to MES. The proposed technique calculates the costs of implementing a component on both MES and local end. In addition, the cost is the offloading decision-dependent variable that finds the perfect offloading procedure for some particular states of a component

(ii) An efficient computation offloading system based upon supervised feed forward architecture has been created, along with random offloading scheme (ROS) and total offloading scheme (TOS) that are employed to evaluate the cost consumption and accuracy rate

(iii) Performance of our proposed strategy shown by numerical simulations, which validate that we achieved the lowest possible cost compared to alternative methods and also observed the lowest slope curve by a parameter constraint mathematical model

The rest of this paper is ordered as follows: Section 2 introduces an outline of the related works. Section 3 discusses the proposed model and methodology followed by experimental results presented in Section 4. Finally, the "Conclusion" section concludes this paper.

2. Related Work

Several methods have been presented to handle mobile offloading problems in dynamic situations. The approaches presented in the literature are based on optimization methods that aid in allocating the MEC resources [17, 18]. Nevertheless, a system based upon MEC is typically quite complex, and occasionally, it is tough to be portrayed in a mathematics arrangement. Similarly, optimization challenge is primarily expressed based upon a snap of the system, then reformulated when the situation fluctuates eventually. Besides, most of the conventional optimization approaches necessitate a hefty number of iterations to seek a local optimum preferably the global optimum.

Even though mobile cloud computing (MCC) attempts to drive restrictions of mobile applications through involving centralized resources to accomplish computational offloading, mobile edge computing (MEC) further proceeds by allocating the key portion of distant operations directly to nearby structures. These resources, characteristically situated at the logical edges of a network, might consist of LTE base stations, where routers deliver joint resources [19]. Mitsis et al. [20] presented a usage-based pricing mechanism and a user's risk-based behavior-aware data offloading decision-making scheme in a UAV-assisted MEC system. Considering the pricing mechanism, prospect theoretic utility functions are formulated to capture users' decision-making behaviors. Moreover, the theory of the tragedy of the commons is utilized to model the UAV's resource utilization. Each participant in the game is expected to maximize its utility function in a noncooperative manner. However, due to the growing number of UAVs, network resource management faces challenges such as power control, spectrum allocation, and task allocation.

The deep learning [12] technique has accomplished its astonishing performance. Deep learning has surpassed techniques based upon machine learning in entire artificial intelligence areas, comprising of speech recognition [21], computer vision [22], natural language processing [23], and so on. Referring to the traditional machine learning-based offloading techniques, namely, the offloading scheme based upon the Markov decision process in [2], the deep learning scheme conveys two superiorities: (i) remarkable accuracy potential of achieving in decision-making and (ii) radical speed of calculation used for the test by a trained model.

Another eminent method of reinforcement learning is suitable for distributed decision-making. In this technique, autonomous agents acquire the most appropriate action by using penalties and rewards obtained during each round of play. Meanwhile, agents are unaware of which action is suitable to take; the agents learn through balancing search of unidentified actions and utilization of the existing data of already utilized actions. Simply put, agents utilize trial and also error tactics to capitalize on their functions over the horizon. Few eminent reinforcement learning approaches comprise learning automata, Q-learning, and Roth-Erev. However, reinforcement learning tactics are quite well suitable for learning in minority game (MG) [24]; subsequently, adjustment to the joint action of other agents in the existence of information deficiency can be accomplished by such approaches [25].

Moreover, in MEC literature, quite a lot of research work is available to mitigate the transmission latency issue and offer more optimized system performance. In [26], an RL framework based upon a deep Q-learning approach was utilized to estimate action-value action. They also provided a strategy to get the overhead-aware optimal computation offloading. Further, each user can learn via surrounding environment interactions and then approximate its performance in value function form. In the next step, the user can choose the overhead-aware whether edge computing or local computing by its condition. Another study also utilized a reinforcement learning algorithm-based technique called deep Q-network [27]. The key finding of their work was to automatically learn the offloading decision to improve the system performance and greatly decrease the latency and energy consumption.

In [28], they studied long-term throughput maximization problems considering a multicell multiuser framework for MEC. They did not solely emphasize two key issues, namely, energy and latency minimization issue, but a novel strategy is presented from the service provider's perception to improve the system-wide throughput with latency limits through equally getting user accord along with resource distribution for communications and computing in consideration. Additionally, the Markov decision process (MDP) is applied to model the queuing conditions for mobile devices and also MEC servers.

In addition to latency and energy issues, computational offloading has risen. In [28], a way to alleviate offloading is presented, that is, distributed deep learning-based offloading (DDLO) algorithm. This approach uses multiple parallel DNNs to produce offloading choices. They follow a joint replay memory to save newly produced offloading choices which are then trained along with improving all DNNs. [29] presented MEC-enabled long-term evolution (LTE) framework and analyzed the impact of numerous vehicular communication modes on the performance of task offloading. They employed extensively used deep Q-learning method for optimal target MEC server decision to assist in maximizing utilities of offloading scheme subjected to specified delay limitations. Further, to improve task offloading reliability suggested an effective redundant offloading algorithm.

Finally, this persuades us to scheme a flexible deep learning-based offloading technique. In our approach, we assume that the mobile users are in a still position when they are performing to offload the mobile task to edge devices, considering that the communication between edge devices and mobile users is always consistent. Therefore, we did not include the mobility of users in our approach. Additionally, the network can become more critical with the user's high mobility.

3. Methodology

The process of implementation of an approach can be distributed into various stages. Every stage with associated data is a component of the approach implementation. The component can be installed either on the mobile edge server or local end. An effective offloading method should choose an optimum component part to offload to MES but not to entire, targeting to which. There are vital stages proposed in this approach, starting with calculating the implementation cost of a component on the MES end and local end, respectively, proving the implementation of the cost function offloading procedure, whereas the value of the cost variable is dependent on the offloading decision, finding the perfect offloading procedure for some particular states of a component with extensive methodology and the perfect offloading procedure including their states of component, respectively, and these two segments are then measured as the inputs and outputs of our training dataset. Lastly, applying a convolutional neural network, with that, we can achieve the perfect offload method of any states of a component through the training dataset. Moreover, Abbreviations represents all the notations to be used in this section.

3.1. Implementation of Local End Cost. The implementation of local end cost contains execution and consumption time. The work in [30] presented that implementation time can be computed through the amount of input data required for a single component. However, this approach neglected that the processed data with input data were not identical in size. Suppose we accept that the component output data is the input data to the incoming component, then we can apply $k_{n-1,n}$ to show the component l_{n-1} input data; thus, the amount of work of component is given as

$$C_l = B \cdot P_n \cdot k_{n-1,n}, \tag{1}$$

where C_l is calculated in the clock cycle of CPU and B shows the amount of clock cycle a microprocessor will process each

byte, and it is computed in cycles per byte. [31] introduced the research of this parameter. P_n is the cost of computation of l_n and denotes the data augmentation feature of l_n. Understandably, the processed data and input data differ in size because a single component might handle the same input data many times; that is the reason the significance of P_n is presented in this work. As a result, if the component l is installed and performed on the local UE end, its implementation time is equivalent to the time required to complete the amount of work C_l and is stated as follows:

$$T_m(l) = \frac{C_l}{f_m}, \tag{2}$$

where is CPU rate for UE denoted by f_m, where million instructions per second (MIPS) are calculated. The energy usage because of this amount of work represented by K_l and energy is given below:

$$K_l = C_l \times V, \tag{3}$$

where V shows the consumption of UE's unit power, and each CPU cycle is calculated in MAH. Assuming the entire energy of UE is K_t, the lingering energy for the coming component $l+1$ can be calculated by

$$K_r = C_l \times (V - K_t). \tag{4}$$

Later, the energy usage and implementation time were computed by the above equations, and the local end implementation component l cost can be measured by the following equation:

$$G_m(l) = \lambda_1 T_m(l) + \lambda_2 K_l, \tag{5}$$

where λ_1 and λ_2 are coefficients of weights which can make equilibrium of energy usage and time delay in the cost function of local end, respectively.

3.2. Implementation of MES End Cost. With the local end implementation, the user equipment can also afford a component offloading to the remote end, such as MES to implement. Similar to the local end, the MES end implementation cost also has the implementation time but is much smaller compared to the local end time. We can assume implementation time same as local end time:

$$T_u(l) = \frac{C_l}{f_n}, \tag{6}$$

where f_n is MES CPU rate. The time used on data transfer from UE to MES has to be kept in mind. This UE mobile Internet environment defines this time; however, at this stage, we have only considered the most regularly applied 4G environment. The orthogonal frequency division multiple access (OFDMA) technology is utilized to deploy the 4G environment. With this kind of access, the download and upload speed relies on the transmission subcarrier number N and bandwidth B. Considering that the similar additive white Gaussian noise (AWGN) channel is broadcast for uplink and downlink, the obtainable uplink and downlink data rate can be simply computed as [32]

$$\begin{aligned} r_u &= n\frac{B}{N}\log_2\left(1 + \frac{p_u|h_{\mathrm{ul}}|^2}{\Gamma(g_{\mathrm{ul}})d^{\beta}N_o}\right), \\ r_d &= n\frac{B}{N}\log_2\left(1 + \frac{p_s|h_{\mathrm{dl}}|^2}{\Gamma(g_{\mathrm{dl}})d^{\beta}N_o}\right), \end{aligned} \tag{7}$$

where B denotes as bandwidth; the path loss exponent is represented by β; the distance between MES and UE is d; further, n is the total subcarriers which are given for from UE to MES transmission; power noise is N_o; p_s and p_u represent the MES and UE transmission power, respectively; the coefficient of channel fading for downlink and uplink is represented by h_{dl} and h_{ul}; and, finally, the required rate of bit error for downlink and uplink is shown by g_{dl} and g_{ul}. Furthermore, the SNR margin for obtaining the bit error rate using quadrature amplitude modulation which is equal to $-2\log 5g_{\mathrm{ul}}/3$ is represented by $\Gamma(g_{\mathrm{dl}})$. Applying the above equation, we can calculate the time taken for UE to dispatch the component l input data to MES can be achieved by

$$T_v(l) = \frac{k_{l-1,l}}{r_u} \times (1 - p_{l-1}), \tag{8}$$

where p_{l-1} describes the offloading decision of the past component $l-1$. The following equation will define the execution of the component:

$$\text{execution} = \begin{cases} \text{MES}, p_{l-1} = 1, \\ \text{Locally}, p_{l-1} = 0. \end{cases} \tag{9}$$

If the past component $l-1$ was done on MES, then the output of that component, for instance, the component l input, requires not to be communicated between MES and UE. Therefore, the time consumed will be zero. On the other hand, the past component $l-1$ was run locally. Therefore, the data communication will be essentially required. Likewise, once the component l execution is completed, the output of that component is likely to be dispatched back to UE if the coming component $l+1$ will be set to execute locally, whereas the communication is not necessary if component $l+1$ will be set to execute on MES; finally, the time consumed for UE to allow the component l output data of MES is given below:

$$T_w = \frac{k_{c,c+1}}{ru} \times (1 - p_{l-1}). \tag{10}$$

Lastly, the cost of MES end implementation is given below:

$$G_n(l) = \lambda_3 T_u(l) + \lambda_4 T_v(l) + \lambda_5 T_w(l), \tag{11}$$

where λ_3, λ_4, and λ_5 are weights that can work as balancing the time, respectively.

3.3. Design a Cost Function. As we all know, a component can perform a task either remotely or locally, where equations (5) and (11) represent the cost function, respectively. To formalize the offloading cost function efficiently, we introduce the single component cost l as

$$\text{cost} = \begin{cases} \text{Remotely}(G_{\text{remote}}), p_l = 0, \\ \text{Locally}(G_{\text{local}}), p_l = 1. \end{cases} \tag{12}$$

The offloading procedure cost function is the sum implemented of all the components and thus can be denoted as

$$G = \sum G_{\text{remote}} + G_{\text{local}}. \tag{13}$$

Suppose the decisions of offloading of all components make the decision environment as $D = d_1, d_2, \cdots, d_M$, whereas M denotes the total number of components used. After, our method is aimed at searching out an authentic decision environment as D^* to minimize the above equation, which we can derive as

$$D^* = \arg_D \min \sum G(l). \tag{14}$$

4. Simulation of Algorithm

To calculate the efficient offloading model as presented in the above equation, a deep ANN structure is done in this article. The most vital stuff is generating the datasets for training for our ANN model. The steps to collect datasets for training are given below:

(i) The decision of component offloading relies on its condition; thus, the condition components and the related decisions of offloading must be the outputs and inputs of the ANN, respectively

(ii) A component possesses a condition (v, c, d, and b) showing the current component mobile environment, where v is denoted as the amount of input data, the component number is denoted as c, the distance between MES and UE is measured as d, and, finally, b shows the bandwidth

(iii) Let us suppose there are total L components, and we arbitrarily produce a condition for every component; therefore, we finally obtain L different conditions. A know possesses two offloading selections; therefore, there would be 2^L distinct offloading states. Applying the exhaustive approach, we computed each state cost and choose among the best which can minimize equation (17)

(iv) After, if the ANN is trained accurately and if we give L states as input, it should give an accurate offloading scheme as output

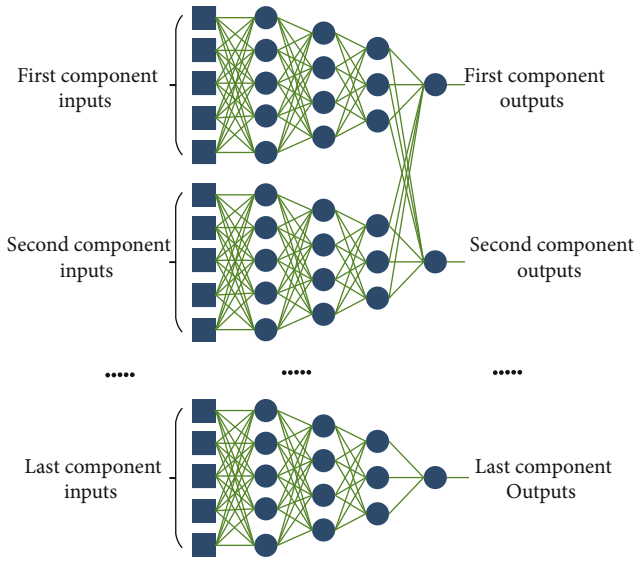


Figure 1: Proposed feed forward neural network.

(v) We train the ANN model for P number of times as done in step number 3; then, we can obtain different conditions and the related better offloading procedures, which make P number of rows data for training. The k-th training data row is shown below:

$$\text{sample}_k = \{S_k, D_k^*\}, \tag{15}$$

where S_k shows the condition of all L components. Hence, S_k is composed of $4L$ data features since each condition consists of 4 condition features. The number of units of the input layer has the same $4L$ to obtain S_k perfectly. D_k^* is the preferred robust L component offload scheme. Expand equation (16) according to the following conditions:

$$\begin{aligned} S_k &= \left\{(v_{k,1}, c_{k,1}, d_{k,1}, b_{k,1}), \cdots\cdots, (v_{k,L}, c_{k,M}, d_{k,M}, b_{k,M})\right\}, \\ D_k^* &= \left(D_{k,1}^*, D_{k,2}^*, \cdots, D_{k,M}^*\right). \end{aligned} \tag{16}$$

For instance, $(v_{k,1}, c_{k,1}, d_{k,1}, b_{k,1})$ is the component 1 condition which is arbitrarily produced in the k-th iteration in step number 5, and $D_{k,1}^*, D_{k,2}^*, \cdots, D_{k,M}^*$ is the related offloading scheme of the same iteration.

The feed forward architecture is shown in Figure 1, which consists of fully connected layers. While we input a training dataset into our ANN, we set a fixed batch size; it will take the first batch as a set of component conditions into it. The training data is generated from a small number of conditions. However, the pretrained ANN is capable of predicting the robust offloading decision of any set of conditions.

Moreover, deep learning is used on offloading policies where computations are managed according to data flow in

the network. Deep learning performs as a smart agent according to the data provided. Data can be lost from low-resource communication channels. However, loss function and stochastic gradient descent can overcome this problem with a loss of negligible accuracy.

4.1. Computational Complexity. The reason for choosing the proposed method is to solve the energy usage by application components, postponements in communication, and computational load, which mainly relies on its benefit of low complexity compared to benchmark solutions, e.g., random offloading and total offloading schemes. In particular, the complexity of random offloading grows exponentially to $O(2^n)$ as the number of fine-grained components n of an application grows, whereas the complexity of the proposed method is typically on the order of $O(mn)^2$, where m is the number of neurons in a hidden layer that indicates the scale of the learning model.

5. Experiment Settings

We performed the experiments with NVIDIA GTX TITAN Xp GPU. We used $L = 100$ and randomly produced $10K$ conditions for every component after we obtained the dataset for training $\{\text{sample}_k = \{S_k, D_k^*\} | k = 1, 2, \cdots, 10K\}$. We implemented different batch sizes of the datasets for training, for instance, 16, 32, 64, 128, 256, 512, and 1020, respectively. The average batch size is 100 MBs. We assume that the mobile users are in a static position while performing to offload the mobile task to edge devices, considering that the communication between edge devices and mobile users is always consistent. Therefore, we did not include the mobility of users in our approach. Additionally, the network can become more critical when user mobility is greater.

We chose three offloading strategies based on the literature and attempted to assess their efficiency; the three schemes are described below:

(i) Random offloading scheme (ROS) [33] randomly chooses robust components irrespective of the volume of data transfer needed, remote and local resources, and network conditions

(ii) Total offloading scheme (TOS) [33] is the coarse-grained method, which transfers the entire computation complexity to MES. This policy does not require a decision on offloading strategies because the entire computations are offloaded

(iii) Offloading policy based on deep learning considers the amount of data transfer needed and network conditions. It applies a deep neural network with 2 middle layers (hidden layers) and 128 units in each layer

Table 1 gives several network parameters applied in this article. Most numbers of parameters applied were similar as in [34], but the rates of CPU (f_n and f_m) were specified according to the theory that MES has a greater CPU rate

Table 1: Model parameters.

Symbols	Value
B	3 [34]
β	-2
d	10 [34]
f_m	10^{10}
f_n	10^5
λ_1	0.5
λ_2	0.5
λ_3	0.5
λ_4	0.3
λ_5	0.1
N_o	0.15×10^5
N	256
p_s	0.1
p_u	0.1
g_{dl}	0.01
g_{ul}	10^{-3}

against the UE. We proposed the predictive performance as follows for comparison purposes:

$$\text{MAE} = \frac{\sum |d_{k,m} - d_{k,m}^*|}{N}, \tag{17}$$

$$\text{RMSE} = \sqrt{\frac{\sum (d_{k,m} - d_{k,m}^*)^2}{N}}, \tag{18}$$

where $d_{k,m}^*$ is a component's predicted offloading policy, d_k is their actual optimal offloading policy, and the total number of components is denoted by the letter N. Point 3 of the above section demonstrated how to obtain the true optimal offloading policy.

Figure 2 depicts our deep model prediction accuracies for various sample values. Figure 2 shows that the model's prediction accuracy improves as the number of samples increases. When the number of samples reaches 50, the root mean square error (RMSE) and mean absolute error (MAE) become less than half, which shows that model can make predictions accurately as $m \le 1, 0 \le d_k$. However, when the sample size approaches a thousand, the prediction accuracy soon deteriorates. It indicates that the model's performance is not proportional to the sparsity. We examined our algorithm's cost consumption and accuracy rate with ROS and TOS. The accuracy rate is formulated as follows:

$$U = \frac{N_p}{M}, \tag{19}$$

where N_p denotes the number of components that have been correctly offloaded. The accuracy rate of our algorithm is examined in Figure 3 between ROS and TOS. The accuracy

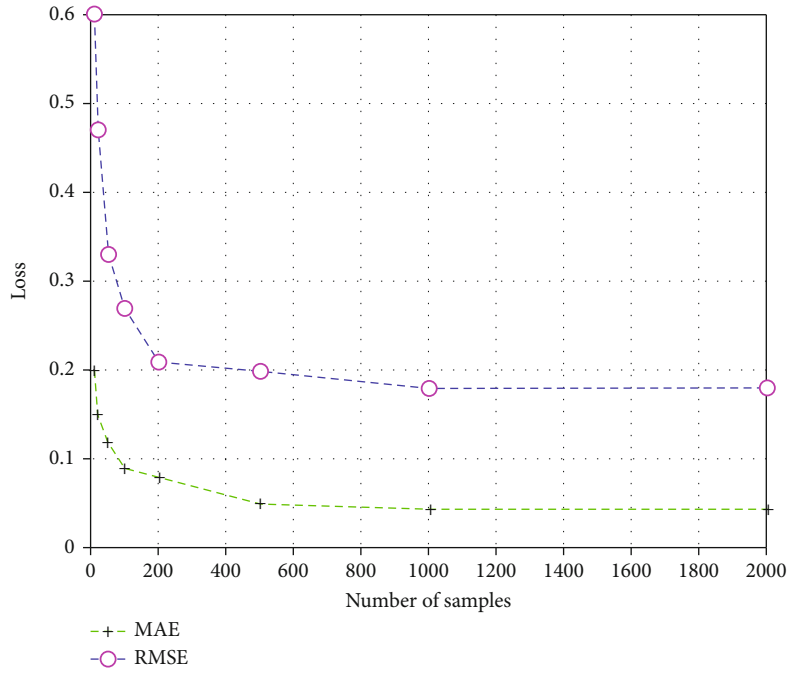


Figure 2: Prediction accuracy of the proposed algorithm.

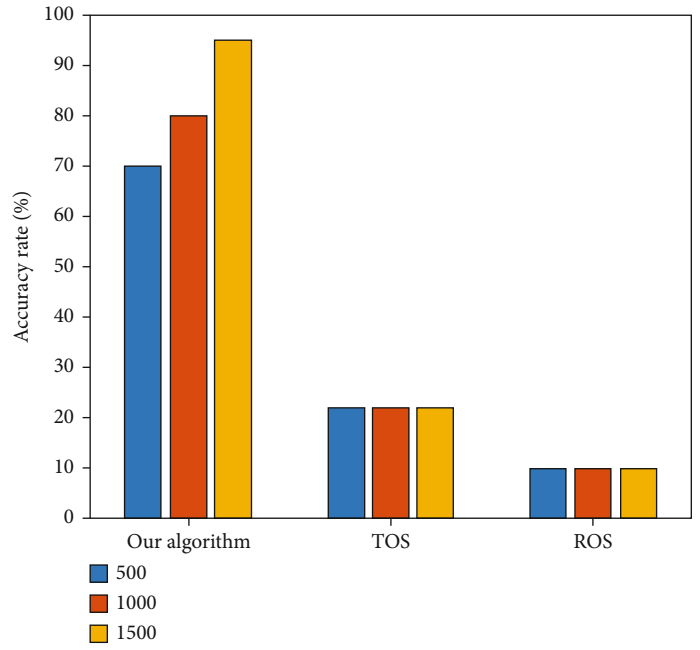
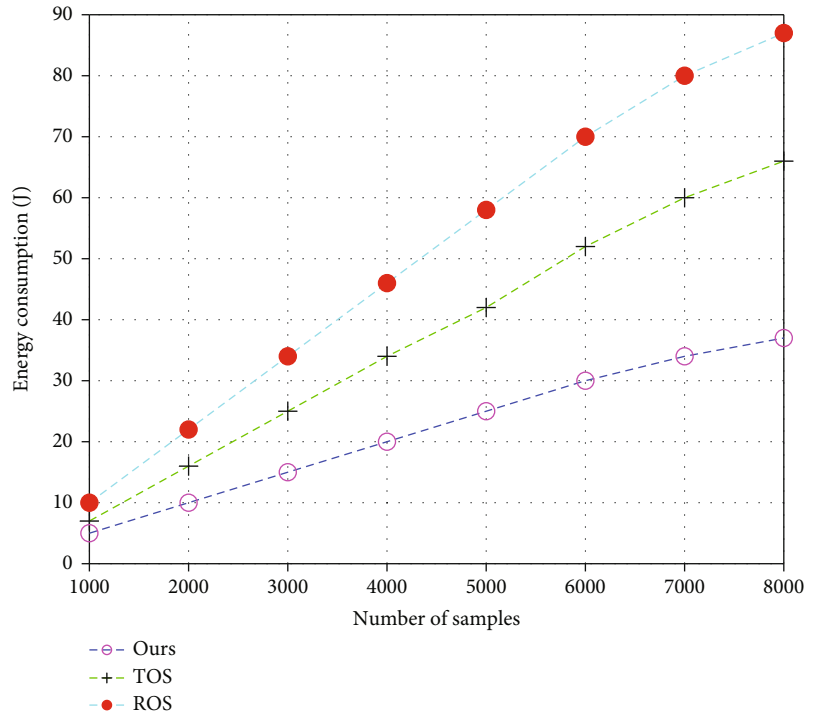


Figure 3: Offloading accuracy rates comparison of the proposed algorithm.

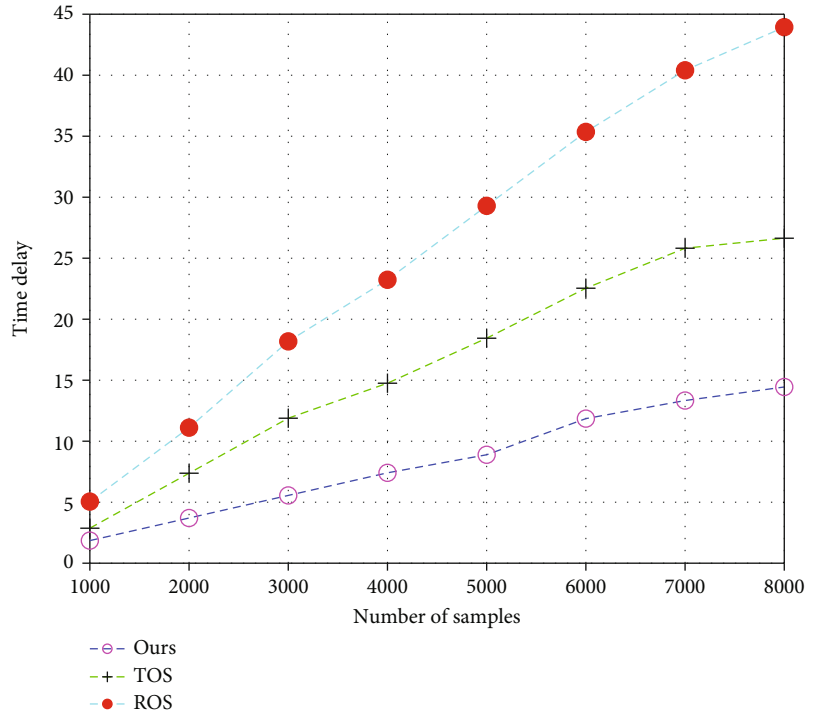
rate is calculated using a training dataset with 500, 1000, 1500 samples, respectively. We found that when the sample size grows larger, our algorithm's prediction accuracy improves. As a result, given a big sample size, the ANN can create a more accurate offloading scheme.

Figure 3 also shows that our proposed algorithm performs better as compared to other schemes. ROS and TOS are offloading either randomly or totally, which lowers the accuracy rates. On the other hand, our proposed scheme improves the accuracy with the help of a deep neural network with various conditions such as energy, data volume, and network condition. This multitude of consideration has significantly improved the overall accuracy under various conditions, as shown in Figure 3.

The performance of the proposed scheme concerning energy and time delay is shown in Figure 4. Figure 4(a) shows that our proposed scheme outperforms TOS and ROS for all number of samples. For the number of samples $= 8000$, our proposed scheme improves the energy consumption by 57% and 87% compared to TOS and ROS,



(a)



(b)

Figure 4: Overall energy consumption and time delay with TOS and ROS.

respectively. Similarly, Figure 4(b) shows the comparison of time consumption of our proposed model with other schemes. The higher accuracy rate of our proposed scheme has also contributed to the lower time delay. It can be observed that our proposed schemes suffers from lower time delay as compared to TOS and ROS and has improved the performance by 26% and 43%, respectively.

Figure 5 depicts the time complexity of our algorithm versus TOS and ROS for different sample sizes. It can be observed from Figure 5 that the time complexity of random

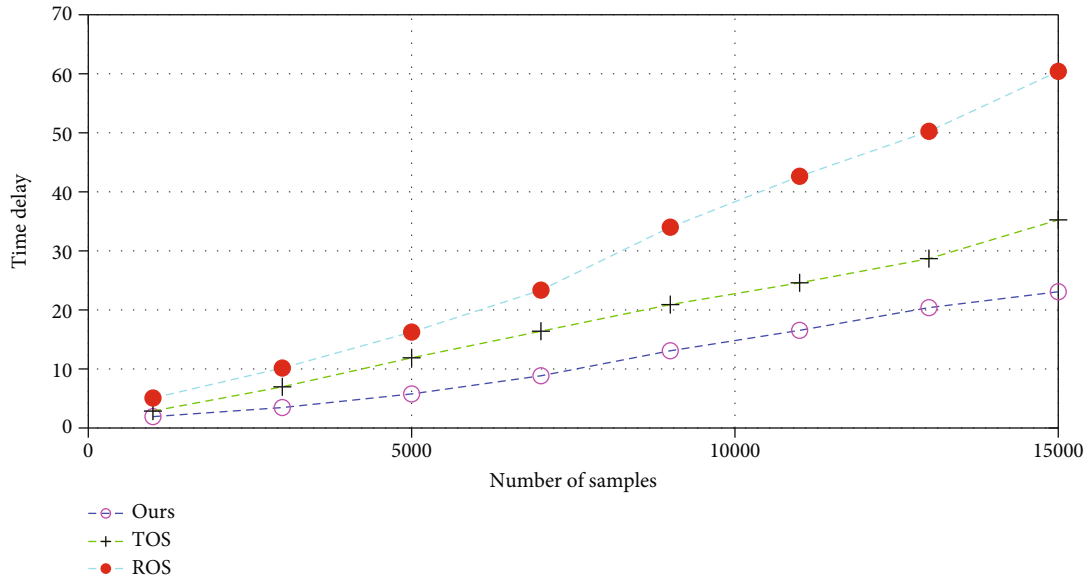


Figure 5: Time delay with TOS and ROS for time complexity.

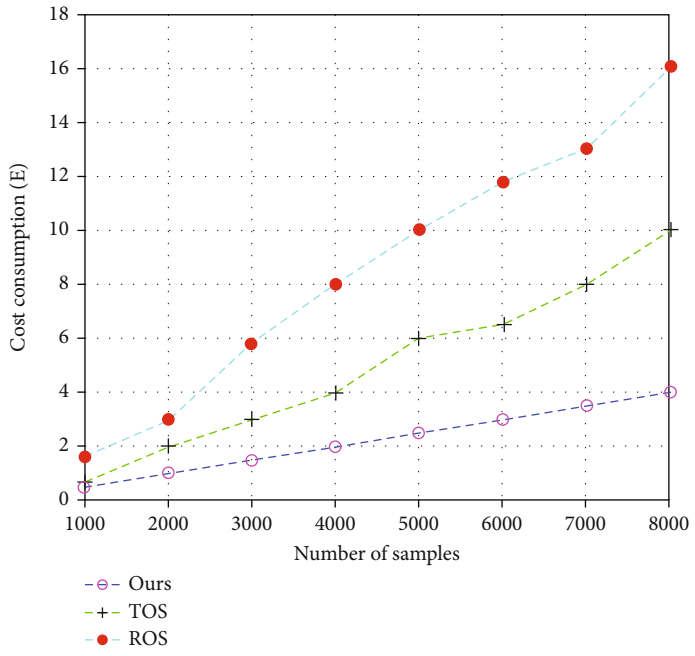


Figure 6: Overall cost comparison of the proposed algorithm with TOS and ROS.

offloading grows exponentially to $O(2^n)$ as the number of samples grows. In contrast, the complexity of the proposed method is typically on the order of $O(mn)^2$, with a slow increase in the graph trend. Therefore, our proposed scheme outperforms the others in terms of learning capacity.

Figure 6 shows the accuracy cost consumption of our algorithm w.r.t. TOS and ROS for various sample sizes. The application will run using our technique at the lowest possible cost when compared to alternative techniques. Another noteworthy point is that our method observed the lowest slope curve, indicating that when the prediction scenario becomes more complicated, our method might significantly decline offloading performance.

The combined energy consumption and time delay effect is depicted in Figure 7, representing λ_1 and λ_2, respectively. We call it *cost of consumption* (E). In addition, to check the impact of energy and time on the performance, the difference values of λ are taken. It is observed that for lower number of samples, energy (λ_1) positively affects E as compared to time (λ_2), which can be seen in Figure 7 for 2000 and 4000 number of samples. We can see the lower cost of consumption (E) for $\lambda_2 > \lambda_1$. However, for the remaining

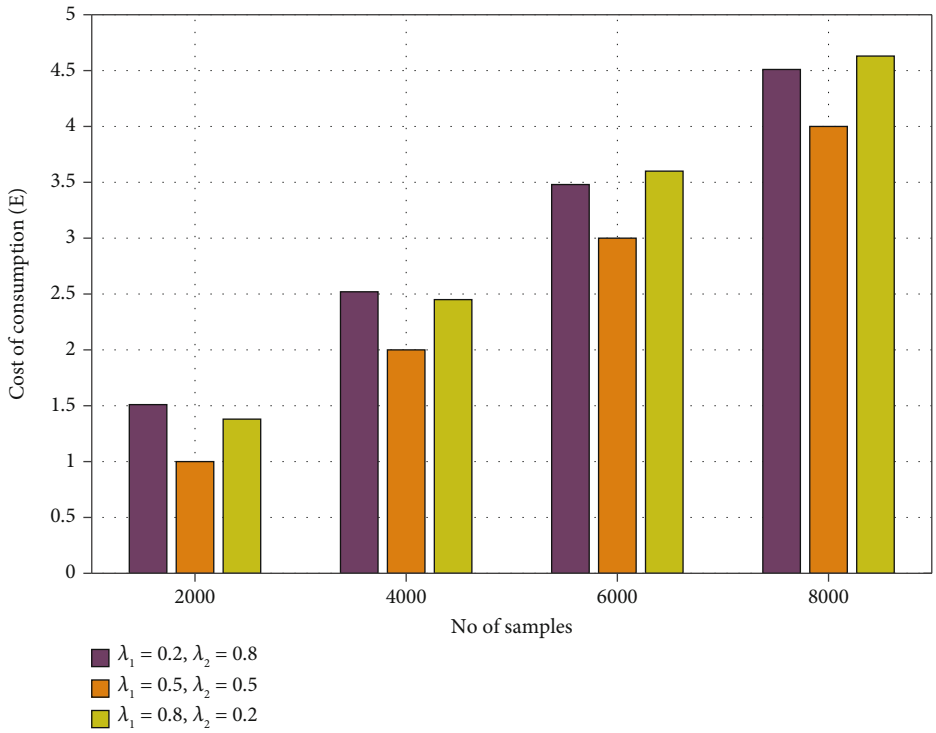


Figure 7: Overall cost of our algorithm with varying λ_1 and λ_2.

samples, it is observed that time (λ_2) plays a significant role in decreasing E as compared to energy (λ_1). It is concluded that both time and energy affect the overall performance and diversely contribute towards the cost of consumption. This unique property can fulfil the user preference related to energy and time. In order to deal with energy constraint applications, higher value of λ_1 can be used. Similarly, for delay intolerant applications, λ_2 can be preferred over λ_1. This way, the users' requirements' variability can be effectively satisfied.

6. Conclusion

In this article, we described a new method to smartly offload application components to the cloud, applying extensive mathematical modeling and deep neural networks technique. We designed a cost function for the application to be implemented on user equipment (UE) as well as on the cloudlet server giving network conditions, accessible computation resources, delays, and energy usage. Considering these factors in the cost function, our introduced method is highly extensive and produces greater accuracy for robust decision-making for the offloading issue in MEC. Throughout a careful measure of the cost function, energy usage, and accuracy, we illustrated that our method is more accurate and uses an advanced state-of-the-art technique. To prevent complex computation and make the process of decision-making quicker, we trained deep neural networks. Further, to train the deep neural networks, the datasets are created from the designed mathematical model in which we include all the significant features in the cost function formula. Our model obtained approximately a 2.3% reduction in the total cost and approximately a 3.1% reduction in energy usage, as distinguished with other past techniques. Finally, we obtained greater accuracy with the lower number of neurons in the deep neural networks.

Abbreviations

C_l: The amount of work of component
$T_m(l)$: Time required to complete the amount of work
K_l: The energy usage on amount of work
λ_1, λ_2: Coefficients of energy and time weights, respectively
r_u: Uplink data rate
r_d: Downlink data rate
G: The offloading procedure cost function.

Data Availability

The data/numbers are generated by the simulation to check validity. Therefore, no supporting dataset is available.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under Grant 61932005 and in part by the 111 Project of China under Grant B16006.

References

[1] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Transactions on Networking*, vol. 24, no. 5, pp. 2795–2808, 2016.

[2] X. Chen, "Decentralized computation offloading game for mobile cloud computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 4, pp. 974–983, 2015.

[3] S. Zhou, W. Jadoon, and J. Shuja, "Machine learning-based offloading strategy for lightweight user mobile edge computing tasks," *Complexity*, vol. 2021, Article ID 6455617, 11 pages, 2021.

[4] Z. S. Khaliq, A. I. Jehangiri, T. Maqsood, J. Shuja, Z. Ahmad, and A. I. Umar, "LiMPO: lightweight mobility prediction and offloading framework using machine learning for mobile edge computing," *Cluster Computing*, vol. 26, no. 1, pp. 99–117, 2023.

[5] M. Maray and J. Shuja, "Mobile cloud computing [Guest Edotorial]," *Mobile Information Systems*, vol. 2022, Article ID 1121822, 17 pages, 2022.

[6] X. Fu, S. Secci, D. Huang, and R. Jana, "Mobile cloud computing [Guest Edotorial]," *IEEE Communications Magazine*, vol. 53, no. 3, pp. 61-62, 2015.

[7] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: a survey," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 450–465, 2018.

[8] R. Roman, J. Lopez, and M. Mambo, "Mobile edge computing, fog et al.: a survey and analysis of security threats and challenges," *Future Generation Computer Systems*, vol. 78, pp. 680–698, 2018.

[9] D. Belli, S. Chessa, L. Foschini, and M. Girolami, "A social-based approach to mobile edge computing," in *2018 IEEE Symposium on Computers and Communications (ISCC)*, pp. 00292–00297, Natal, Brazil, 2018.

[10] S. Wang, X. Zhang, Y. Zhang, L. Wang, J. Yang, and W. Wang, "A survey on mobile edge networks: convergence of computing, caching and communications," *IEEE Access*, vol. 5, pp. 6757–6779, 2017.

[11] L. Yang, H. Zhang, M. Li, J. Guo, and H. Ji, "Mobile edge computing empowered energy efficient task offloading in 5G," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 7, pp. 6398–6409, 2018.

[12] J. Zhang, X. Hu, Z. Ning et al., "Energy-latency tradeoff for energy-aware offloading in mobile edge computing networks," *IEEE Internet of Things Journal*, vol. 5, no. 4, pp. 2633–2645, 2018.

[13] Z. Tong, X. Deng, F. Ye, S. Basodi, X. Xiao, and Y. Pan, "Adaptive computation offloading and resource allocation strategy in a mobile edge computing environment," *Information Sciences*, vol. 537, pp. 116–131, 2020.

[14] J. Xu, X. Li, R. Ding, and X. Liu, "Energy efficient multi-resource computation offloading strategy in mobile edge computing," *Jisuanji Jicheng Zhizao Xitong/Computer integrated manufacturing systems, CIMS*, vol. 25, no. 4, pp. 954–961, 2019.

[15] X. Zhao, J. Peng, and W. You, "A privacy-aware computation offloading method based on Lyapunov optimization," *JEIT*, vol. 42, no. 3, pp. 704–711, 2020.

[16] L. Liu, X. Liu, S. Zeng, T. Wang, and R. Pang, "Research on virtual machines migration strategy based on mobile user mobility in mobile edge computing," *Journal of Chongqing University of Posts and Telecommunications*, vol. 31, pp. 158–165, 2019.

[17] T. Q. Dinh, J. Tang, Q. D. La, and T. Q. S. Quek, "Offloading in mobile edge computing: task allocation and computational frequency scaling," *IEEE Transactions on Communications*, vol. 65, no. 8, pp. 1–3584, 2017.

[18] P. Mach and Z. Becvar, "Mobile edge computing: a survey on architecture and computation offloading," *IEEE Communications Surveys Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.

[19] Cisco DevNet, *Cisco DevNet*, 2022, January 2022, https://developer.cisco.com.

[20] G. Mitsis, E. E. Tsiropoulou, and S. Papavassiliou, "Data offloading in UAV-assisted multi-access edge computing systems: a resource-based pricing and user risk-awareness approach," *Sensors*, vol. 20, no. 8, p. 2434, 2020.

[21] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz, and J. Silovsky, Eds., "The Kaldi speech recognition toolkit," in *IEEE 2011 workshop on automatic speech recognition and understanding*, IEEE Signal Processing Society, 2011.

[22] D. Forsyth and J. Ponce, *Computer Vision: A Modern Approach*, Prentice Hall, Second edition, 2011, January 2022, https://hal.inria.fr/hal-01063327.

[23] G. G. Chowdhury, "Natural language processing," *Annual Review of Information Science and Technology*, vol. 37, no. 1, pp. 51–89, 2003.

[24] S. Ranadheera, S. Maghsudi, and E. Hossain, "Minority games with applications to distributed decision making and control in wireless networks," *IEEE Wireless Communications*, vol. 24, no. 5, pp. 184–192, 2017.

[25] D. Catteeuw and B. Manderick, "Heterogeneous populations of learning agents in the minority game," in *Adaptive and Learning Agents*, pp. 100–113, Springer, Berlin, Heidelberg, 2012.

[26] Y. Wei, Z. Wang, D. Guo, and F. R. Yu, "Deep q-learning based computation offloading strategy for mobile edge computing," *Computers, Materials and Continua*, vol. 59, no. 1, pp. 89–104, 2019.

[27] R. Zhao, X. Wang, J. Xia, and L. Fan, "Deep reinforcement learning based mobile edge computing for intelligent Internet of Things," *Physical Communication*, vol. 43, article 101184, 2020.

[28] Y. Deng, Z. Chen, X. Chen, and Y. Fang, "Throughput maximization for multiedge multiuser edge computing systems," *IEEE Internet of Things Journal*, vol. 9, no. 1, pp. 68–79, 2022.

[29] K. Zhang, Y. Zhu, S. Leng, Y. He, S. Maharjan, and Y. Zhang, "Deep learning empowered task offloading for mobile edge computing in urban informatics," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 7635–7647, 2019.

[30] G. Orsini, D. Bade, and W. Lamersdorf, "CloudAware: a context-adaptive middleware for mobile edge and cloud computing applications," in *2016 IEEE 1st International Workshops on Foundations and Applications of Self* Systems (FAS*W)*, pp. 216–221, IEEE, Augsburg, Germany, 2016.

[31] M. Yang, Y. Wen, J. Cai, and C. H. Foh, "Energy minimization via dynamic voltage scaling for real-time video encoding on mobile devices," in *2012 IEEE International Conference on Communications (ICC)*, pp. 2026–2031, Ottawa, ON, Canada, 2012.

[32] X. Ran, H. Chen, Z. Liu, and J. Chen, "Delivering deep learning to mobile devices via offloading," in *Proceedings of the Workshop on Virtual Reality and Augmented Reality Network*, pp. 42–47, New York, NY, USA, 2017.

[33] S. Raza, W. Liu, M. Ahmed et al., "An efficient task offloading scheme in vehicular edge computing," *Journal of Cloud Computing*, vol. 9, no. 1, pp. 1–14, 2020.

[34] S. Yu, X. Wang, and R. Langar, "Computation offloading for mobile edge computing: a deep learning approach," in *2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, pp. 1–6, Montreal, QC, Canada, 2017.