كلية الهندسة بشبرا
FACULTY OF ENGINEERING AT SHOUBRA
ERJ
ISSN:1687-1340

ENGINEERING RESEARCH JOURNAL (ERJ)
Volume (52),Issue (2)
April 2023, pp:51-63
https://erjsh.journals.ekb.eg

Comparison of different strategies for Time and Energy Efficient Offloading for Mobile Edge Computing

Rania Azouz [1,*], Esraa Mosleh Eid [1], Lamiaa Elrefaei [1], Heba A. TagElDien [1]

[1] *Department of Electrical Engineering, Faculty of Engineering, Shoubra, Benha University, Cairo ,Egypt.*
** Corresponding author*

E-mail address: rania.salim16@feng.bu.edu.eg, esraa.soliman@feng.bu.edu.eg, lamia.alrefaai@feng.bu.edu.eg, hebaallah.shahat@feng.bu.edu.eg

Abstract: Every day, the number of wireless devices, and IoT applications increases, which require extensive computational resources. Therefore, it is possible to mitigate the lack of computational resources in wireless devices by using Mobile Edge Computing (MEC). MEC is a modern technology that brings the capabilities of Cloud Computing at the edge of a mobile network to perform computationally intensive tasks, which reduces the delay and prevents end to end communication with the remote Cloud. This paper proposed a task offloading model for multiple-device, multiple-task MEC system, the model is formulated as an optimization problem with the objective of reducing time of computation and energy consumption. However, the complexity rapidly increases as more devices are added to the system, thus the proposed problem is solved by introducing five strategies which are full local computing, full offloading computing, random offloading, Q learning, Deep Q network, and a distributed DNN, which are compared with the optimal offloading strategy. The results (4 devices with 3 tasks for each device) show that the total cost in terms of time and energy consumption in Q learning, DQN and, Distributed DNN algorithms is near to the optimal offloading strategy, furthermore, these strategies reduce the total cost up to 63.7% when compared to full local strategy, also up to 21.8% when compared to full edge strategy. However, the learning speed of distributed DNN is faster than Deep Q Network, when number of devices increases. In addition, adistributed DNN generates the offloading decision (in 4 milliseconds) faster than DQN algorithm (in 8 milliseconds).

Keywords: Deep learning, Task offloading, Resource allocation, MEC, OFDMA.

1. Introduction

In the past, mobile phones was limited to making calls and sending text messages. With the rapid development of mobile networks and technology, the smart phones develop for use in a variety of applications, such as Face Recognition, Augmented Reality, and Virtual Reality [1][2]. Although, the smart phones and wireless devices have computational resources to run these applications, they are unfit to work efficiently due to most of these applications require computationally intensive resources [3][4]. To mitigate the resource limitations of wireless devices, computationally intensive tasks can be offloaded to other resourceful devices. Hence, the concept of computation offloading was born [5].

Mobile Cloud Computing is considered popular method which used in offloading, where the applications or tasks are offloaded to the cloud with intensive resources through the wireless channel to reduce load and extend battery life of wireless devices [6]-[8]. However, because of the long distance between wireless devices and cloud servers, offloading to the cloud requires significant latency and places an additional burden on mobile network. So, it is not suitable to execute real time applications.

Mobile Edge Computing (MEC) is considered an effective solution to address the problems associated with Mobile Cloud Computing. In MEC, Cloud services and resources are being placed near wireless devices at the network's edge, at locations such as Wi-Fi, access points, or the Base Station (BS) [9]. So, MEC can provide low latency and high bandwidth, which enables to execute real time applications [10][11]. Many studies use the principle of task offloading on the MEC to reduce the energy consumption, efficiently allocate the resources, minimize the time of computation, maximize the system utility, and reduce the total cost of wireless device. However, getting the optimal offloading decision in multiple-device and Multiple-task MEC system is a great challenge. Here, Deep learning and Machine learning can be used as efficient techniques for offloading decision [12].

Deep learning algorithms based on Neural Network are a relatively recent field research that has seen great improvement in the last decade and will continue to develop as the Deep learning became more able to deal with more complex problem. Hence, Deep Q network and a distributed Deep Neural Network-based task offloading algorithms is developed to generate the optimal offloading decision [13]. The summery of the main contributions in this paper are as following:

- Task offloading and resource allocation model is formulated as an optimization problem with the objective of reducing the total cost in terms of time of computation and energy consumption for multiple-device, multiple-task MEC system.
- The optimization problem is modified to an equivalent form of deep learning techniques such as Deep Q network (DQN) and distributed Deep Neural Network (DNN)-based task offloading to solve the problem

- The simulation results show that the total cost in terms of time and energy consumption in DQN and, Distributed DNN algorithm is near to the optimal offloading strategy and reduce the total cost up to 63.7% when compared to full local strategy, also up to 21.8% when compared to full edge strategy. In addition, a distributed DNN are used to determine the best policy of accelerate learning.

The rest of this paper is organized as follows. Section 2 is the related work of computation offloading to MEC server. Section 3 includes the mathematical operation which used to solve the problem. Section 4 includes the implemented strategies. Section 5 shows the simulation and the results of our model. Section 6 shows the conclusion of this paper and the future work.

2. RELATED WORK

In recent year, computation offloading in MEC systems has become an important research topic in many scientific papers due to increasing number of wireless devices and IoT applications which required an extensive computation and resources. So, in this section, we will mention the previous studies about task offloading for edge computing. In section 2.1, we will mention the studies, which use the traditional optimization methods to solve the problem. Recently, there are many studies, which solved the problem by using machine learning and deep learning, as in section 2.2. The computation offloading may be binary or partial offloading to the edge server. where binary offloading is meaning that the whole data size is offloaded to the edge server or locally execute at a wireless device, while the data size is divided where a subset of the data is executed locally or remotely, this is meaning a partial offloading [14]. In this study, we executed a binary computation offloading. Some of computation offloading studies are categorized by number of MEC servers, users, and tasks. In [19][24][25] computation offloading model is single user. Others study multi users, multi tasks, and multi MEC servers [21].

2.1 Traditional optimization methods

The main objectives of task offloading minimize required time to execute the task, and energy consumption of wireless devices. Many of studies used the traditional optimization method to achieve these objectives [15][16]. Zhang et al. [17] used game theory to solve the optimization problem of task offloading and resource allocation to minimize the latency and energy consumption. The main goal of Mao et al. [18] is minimizing the latency and task failure. The author used the Lyapunov optimization to achieve this goal. However, they did not consider the energy consumption due to the energy harvesting is used as renewable energy. Salmani et al. [19] applied the optimization method to minimize the energy consumption where the offloading may be partial or binary offloading for independent tasks. Elgendy et al. [20][21] joint security, computation offloading, and resource allocation, in addition the author used optimization method to solve the problem, where the author in [18] assumed that the model has single task. In addition, the objectives are to minimize the latency and energy consumption. The extension of [20] is [21] where the model has multi tasks, where the transmitted data was be compressed to minimize the latency. Wan et al. [22] used NOMA to offload the task to MEC server, the objective is minimizing the total delay of the computation of all tasks, they used a heuristic algorithm to solve the optimization problem. The summary of traditional optimization methods is mentioned in TABLE 1, as it shows the open research areas and the drawbacks in each paper.

TABLE 1: Summary of traditional optimization methods

Reference	Proposed method	Algorithm	Objective	Devices	Tasks	Servers	Security
Zhang 2015 [17]	A distributed joint computation offloading and resource allocation optimization method	Game theory	Minimize the latency and energy consumption	Multiple	Multiple	Single	No
Mao 2016 [18]	investigate a green MEC system with energy harvesting devices and develop an effective computation offloading strategy	Lyapunov optimization	Minimize the latency	Single	No	Single	No
Salmani 2020 [19]	address the uplink communication resource allocation for offloading systems that exploit the full capabilities of the multiple access channel.	Optimization method	Minimize the energy consumption	Multiple	Single	Single	No
Elgendy 2019 [20]	A resource allocation and computation offloading model with data security	Optimization method	Minimize energy and latency	Multiple	Single	Single	Yes
Elgendy2020 [21]	A Multi-User Multi-Task Computation Offloading model	Optimization method	Minimize the cost	Multiple	Multiple	Single	Yes
Wan 2021 [22]	NOMA-based multi-access MEC system with multiple MEC servers and multiple users	a heuristic algorithm	Minimize the delay	Multiple	Multiple	Multiple	No

2.2 Deep learning algorithms

Simultaneously, Machine learning and Deep learning have been widely applied in a variety of MEC offloading fields. Due to the benefits of no previous information and low complexity [12][23]. Ali et al. [25] minimize the total cost by using an energy efficient and faster deep learning based offloading technique (EFDOT). The author model has a single user with a single task where the task is divided it into optimal numbers of components and each component either execute inside the local device or offloading to edge server. In [26], the author used edge computing and cloud computing to minimize the total cost in terms of latency and energy consumption of autonomous vehicles. The author used parallel deep neural networks to calculate the optimal offloading decision.

Recently, deep reinforcement learning is used to make computing offloading [27][28]. Li et al. [27] used Deep Deterministic Policy Gradient (DDPG) algorithm to minimize the delay, there are multiple tasks, where each task selects the subnet edge according to type of tasks. Huang et al [28] used the deep reinforcement learning to maximize the computation rate. Elgendy et al. [29] used Q learning and deep Q learning to solve the optimization problem and calculate the optimal offloading decision which reduce the latency and energy consumption. When the task is transferred to the edge server to compute it through the wireless channel, the attackers may attack the task, so some of papers takes in calculations the security [30][31]. In [30], before the task transmit to the edge, the task is encrypted to reduce the attacks. a standard symmetric cryptography algorithm (AES) is used to encrypt task. In [32], the author was used deep supervise learning to find the offloading decision and bandwidth allocation. This algorithm trains faster, but it needs the labels data. Moreover, he used single task for each device and single server. In [33], the author discussed the issue of service migration when hosting multiple users and multiple edge servers to decide to move a continuous service from the edge server to other edge server, so he proposed a Deep Recurrent Q Network-based service migration decision algorithm (DRQNSM) to reduce time and energy consumption while making sure reliable, stable, and continuous services during user movement. To compare DRQNSM with the classic reinforcement learning. In [34], the author focused on offloading of tasks with varying priorities for multi-device, multi-task MEC system. Hence, the author proposed a double reinforcement learning computing offloading (DRLCO) algorithm which makes the decision on offloading, transmission power, and CPU frequency to reduce energy and time. The summary of the deep learning algorithms is mentioned in TABLE 2, as it shows the open research areas and the drawbacks in each paper.

TABLE 2: Summary of deep learning algorithms

Reference	Proposed method	Algorithm	Objective	Devices	Tasks	Servers	Security
Ali 2021 [25]	Faster deep learning based offloading technique to minimize the overall cost	Deep Learning	Minimize the cost	Single	Single	Single	No
Khayyat 2020 [26]	Distributed Deep Learning algorithm used to find the optimal offloading decision	Distributed Deep Learning algorithm	Minimize the total cost	Multiple	Single	Multiple	Yes
Li 2020 [27]	Deep reinforcement learning was used to solve the difficult computation offloading issue	Deep Deterministic Policy Gradient (DDPG)	Minimize delay	Multiple	Multiple	Multiple	Yes
Huang 2019 [28]	Deep reinforcement learning was used for the online offloading decision	Deep reinforcement learning-based online offloading (DROO)	Maximize computation rate	Multiple	Single	Single	No
Elgendy 2021 [29]	Joint computation offloading and task caching	Q Learning, Deep Q Network	Minimize the cost	Multiple	Multiple	Single	No
Elgendy2021 [30]	Security-aware data offloading and resource allocation model	Deep Q Network	Minimize time and energy	Multiple	Multiple	Single	Yes
Chen 2021 [31]	A novel service migration scheme to support mobility	Deep Q Network, Double Deep Q Network	Minimize the cost	Multiple	Single	Multiple	No
Yang 2022 [32]	Deep supervised learning-based computational offloading algorithm (DSLO)was used for computational tasks in MEC networks	Deep learning	Minimize the system utility of MEC network	Multiple	single	single	No

Chen 2023 [33]	Deep Recurrent Q Network-based service migration decision algorithm (DRQNSM) to reduce time and energy consumption	Deep learning	Minimize time and energy consumption	Multiple	single	Multiple	No
Liao 2023 [34]	double reinforcement learning computing offloading (DRLCO) algorithm which makes the decision on offloading, transmission power, and CPU frequency	Deep Reinforcement learning	Minimize time and energy	Multiple	Multiple	single	No

From the above studies, we used six strategies to decrease the total cost in terms of energy consumption and latency then compare between them through the learning and the testing speed, where the model has multiple users and multiple tasks.

3. SYSTEM MODEL

We begin by assuming that there are *N* wireless devices, as indicated by

$$\mathcal{N} = \{1, 2, \dots, N\} \quad (1)$$

where each device has independent M tasks, which can be executed locally or by offloading to a MEC server, as indicated by

$$\mathcal{M} = \{1, 2, \dots, M\} \quad (2)$$

In addition, as shown in Fig.1, there are a Base Station with a MEC server, and a wireless devices connected to a Base Station via a wireless channel. Furthermore, the band width will be equally shared between the wireless devices which decide to offload the tasks.

The decision of offloading for task n of user m is denoted by $z_{n,m} \in \{0,1\}$, as shown in eq. (3), where $z_{n,m} = 0$ means that the device n decides to execute its task m locally, and $z_{n,m} = 1$ means that the user n decides to offload its task m to a MEC server.

$$z_{n,m} = \begin{cases} 0 & local\ excution \\ 1 & remote\ execution \end{cases} \quad (3)$$

The next subsection will explain the computation model and the optimization problem by details.

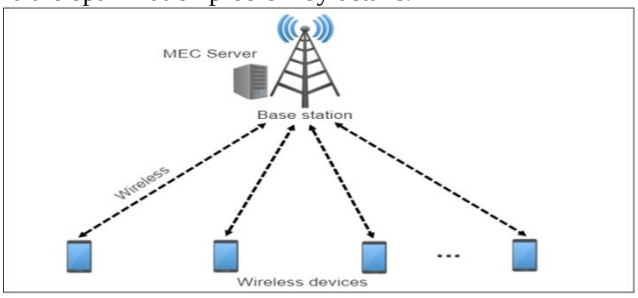

Fig. 1: System model architecture [20]

3.1 Computation model

In the following section, we will discuss the formulas used to determine the computation time, energy consumption and cost used in varying strategies.

3.1.1 Local computing model

In the local computing, the user n selects to compute its task m locally by using its computing resources and the time required to compute this task locally can be calculated by $T^l_{n,m}$ according to eq. (4):

$$T^l_{n,m} = \frac{C\, S_{n,m}}{C^l_n} \quad (4)$$

Where C^l_n is the CPU frequency (cycle/s) of wireless device n, $S_{n,m}$ is the size of task m of user n (bit), and C number of cycles required to execute one bit.

The energy required to compute the task m of user n according to eq. (5):

$$E^l_{n,m} = \beta_n\, C\, S_{n,m} \quad (5)$$

Where β_n is the consumed energy of user n per CPU cycle, $\beta_n = 10^{-27}(C^l_n)^2$ [44].

The following formula can be used to determine the total cost of local computing:

$$Cost^l_{n,m} = k^t_n T^l_{n,m} + k^e_n E^l_{n,m} \quad (6)$$

which is derived from eq. (4) and eq. (5), where k^t_n and $k^e_n \in [0,1]$ are the weights of time and energy consumption, respectively. Where the sum of two weights equals 1. If $k^e_n = 1$ and $k^t_n = 0$, this means that the energy consumption is more sensitive than the time. If $k^e_n = 0$ and $k^t_n = 1$, this means the time is more sensitive than the energy, especially when the existing application is real-time application such as online gaming. So, the values of weights are set according to the application required to execute.

3.1.2 Edge computation model

In Edge computation, first, the user n selects to offload its task m to the Base Station via the wireless channel, then execute it in MEC server. The uplink data rate is required to offload the task to MEC server is calculated according to eq. (7):

$$r_n = B\ \log_2\left(1 + \frac{p_n H_n}{B N_0}\right) \quad (7)$$

Where, B is the total bandwidth of a wireless channel, shared between all wireless devices select to offload, N_0 is the density of noise power, and p_n is the transmission power of device n. The radio resource allocation at the same cell is based on orthogonal frequency division

multiple access (OFDMA) where orthogonal frequency prevents or reduces interference between uplink channel [21][26], H_n is channel gain which follows Rician fading distribution, which determined by eq. (8) according to [35], Where $\bar{H}_n$ is the average channel gain which determined by eq. (9):

$$H_n = 0.3\ \bar{H}_n \tag{8}$$

$$\bar{H}_n = A_d \left(\frac{3 \times 10^8}{4\pi f_c d_i}\right)^{l_e} \tag{9}$$

Where A_d= 3 gives the antenna gain, 3 x 10^8 is the speed of light, f_c=915 MHZ gives carrier frequency, d_n= 120 + 15(n - 1) , for n = 1, · · · , N, where d_n is the distance between wireless device n and Base Station, l_e is the path loss exponent.

The transmission time required to upload the task to the Base Station is calculated according to eq. (10):

$$T^t_{n,m} = \frac{s_{n,m}}{r_n} \tag{10}$$

After the task offload to Base Station, MEC server begin to execute the task by using the computation resources of MEC server. Where the time required to execute the task in MEC server is determined according to eq. (11):

$$T^{exec}_{n,m} = \frac{C\, s_{n,m}}{C^s_n} \tag{11}$$

Where C^s_n is the CPU frequency (cycle/sec) of MEC server that is allocated to device n, it is determined by divided the CPU frequency of MEC server F on number of offloading devices. Furthermore, the time required to download the executed task from MEC server to wireless device is neglected due to the data size of the executed task is small. The total time from offloading to executing the task is determined according to eq. (12):

$$T^s_{n,m} = T^t_{n,m} + T^{exc}_{n,m} \tag{12}$$

Where $T^t_{n,m}$ and $T^{exc}_{n,m}$ are determined by eq. (10), (11) respectively. Energy consumption is needed to offload the task to MEC server is defined as $E^s_{n,m}$:

$$E^s_{n,m} = \mathrm{p}_n\, T^t_{n,m} \tag{13}$$

Where p_n is the transmitted power (watt).

The total cost to offload the user's n task m is determine according to eq. (14):

$$Cost^s_{n,m} = k^t_n T^s_{n,m} + k^e_n E^s_{n,m} \tag{14}$$

Where k^t_n and k^e_n are the weights, which mentioned in the local computing.

3.2 Optimization Problem

The objective of our model is to decrease the total cost of time and energy consumption, where the total cost of all tasks is determined according to eq. (15):

$$Cost^{total} = \sum_{n=1}^{N}\sum_{m=1}^{M}\left(1 - z_{n,m}\right) Cost^l_{n,m} + z_{n,m}\, Cost^s_{n,m} \tag{15}$$

Where if the user n selects to offload its task m, the offloading decision is set to $z_{n,m} = 1$. Otherwise, if the user n selects to locally execute its task m, the offloading decision is set to $z_{n,m} = 0$.

The offloading decision as optimization problem to decrease the total cost is considered according to:

$$\mathrm{Cost}^* = \min_{z}\left[\sum_{n=1}^{N}\sum_{m=1}^{M}\left(1 - z_{n,m}\right) Cost^l_{n,m} + z_{n,m}\, Cost^s_{n,m}\right]$$

$$\text{s.t} \quad \left[E_{n,m} - E^l_{n,m}\right] \le 0 \qquad \text{C1}$$

$$\textstyle\sum_{n=1}^{N}\sum_{m=1}^{M} z_{n,m}\, r_n \le R \qquad \text{C2}$$

$$\textstyle\sum_{n=1}^{N}\sum_{m=1}^{M} z_{n,m}\, C^s_n \le F \qquad \text{C3}$$

$$z_{n,m} \in \{0,1\}, \quad \forall_{n,m} \qquad \text{C4} \tag{16}$$

Where C1 refers to the required energy to execute the task remotely is less than the required energy to execute the task locally. C2 shows that when the users decide to offload their tasks must the sum of data rate of all wireless devices not exceeding the total uplink data rate R. C3 forbids that the sum of computation resources of MEC server to wireless devices to execute the tasks remotely must not be greater than the total available computation resources of MEC server F, where F is CPU frequency of MEC server.C4 shows that the offloading decision is binary where each wireless device only choose to execute the tasks locally or remotely.

4. STRATEGIES

In this paper, there are six strategies applied to execute the tasks. We will explain each strategy in detail as seen below.

4.1 Full local computing strategy

Full local computing, Algorithm 1, is meaning that all wireless devices select to execute all tasks locally, where the offloading decision is set to $z_{n,m} = 0$, such that the offloading decision vector is $z = [0,0,\dots,0]$.We begin to initialize the input parameters used to calculate the time ,eq.(4), the energy consumption, eq.(5), and the cost of each task, eq. (6), then sum the cost of all tasks to find the total local cost.

4.2 Full edge computing strategy

The full edge computing strategy, Algorithm2, is meaning that all users select to execute their task at a MEC sever, where the offloading decision is $z_{n,m} = 0$, such that the offloading decision vector is $z = [1,1,\dots,1]$. Then we initialize the input parameters such that size of computed data $S_{n,m}$, a CPU cycles of MEC server F, the transmitted

power p_n, the channel band width B.W, …. We begin to determine the channel gain of each device according to eq. (8) and (9) to determine uplink date rate of each device according to eq. (7), where the band width of the wireless channel is shared equally between the devices. Then we determine the time and the energy consumption for each task of the user according to eq. (12) and (13) respectively. Note that the resources allocation of MEC server F are divided equally between all tasks.in the end, calculate the cost of each task then sum together to calculate the total cost.

Algorithm 1: Full local computing strategy

Input: N, M, $S_{n,m}, C_n^l, \beta_n, k_n^t, k_n^e$

Output: $Cost^{total}$

1: $Cost^{total} \leftarrow 0$

2: $for\ n = 1, 2, \dots, N$

3: $for\ m = 1, 2, \dots, M$

4: Calculate time $T_{n,m}^l$ according to eq. (4)

5: Calculate energy consumption $E_{n,m}^l$ according to eq. (5)

6: Calculate the cost $Cost_{n,m}^l$ according to eq. (6)

7: $Cost^{total} \leftarrow Cost^{total} + Cost_{n,m}^l$

8: return $Cost^{total}$

Algorithm 2: Full edge computing strategy

Input: N, M, r_n, B, N_0, F, $S_{n,m}$, p_n, k_n^t, k_n^e

Output: $Cost^{total}$

1: $Cost^{total} \leftarrow 0$

2: $for\ n = 1, 2, \dots, N$

3: $for\ m = 1, 2, \dots, M$

4: Allocate the resource allocation $according\ to\ C_n^s \leftarrow \frac{F}{N}$

5: Calculate channel gain according to eq. (8), (9)

6: Calculate uplink rate according to eq. (7)

7: Calculate time $T_{n,m}^s$ according to eq. (12)

8: Calculate energy consumption $E_{n,m}^s$ from eq. (13)

9: Calculate the cost $Cost_{n,m}^s$ according to eq. (14)

10: $Cost^{total} \leftarrow Cost^{total} + Cost_{n,m}^s$

11: return $Cost^{total}$

4.3 Random offloading strategy

Random offloading strategy, Algorithm 3, is meaning that each device selects randomly to execute each task locally or offloading to MEC server, where the offloading decision can be composed from a variety of ways, such as $z = [1, 1, 0, 0, \dots, 0]$.Then calculate the total cost $\text{Cost}^{\text{total}}$. Finally, return the best total cost $\text{Cost}_{\text{min}}^{\text{total}}$.

Algorithm 3: Random offloading strategy

Input: N, M

Output: $Cost_{min}^{total}$

$Cost_{min}^{total} \leftarrow \infty$

1: $\boldsymbol{for}\ sample$

2: Generate random $offloading\ decision\ z$

3: Calculate $Cost^{total} according\ to\ Algorithm\ (4)$

4: if $Cost^{total} < Cost_{min}^{total}$ to calculate the best total cost

5: $Cost_{min}^{total} \leftarrow Cost^{total}$

6: **return** $Cost_{min}^{total}$

Algorithm 4: Total Cost

Input: N, M, r_n, C_n^l, β_n, B, N_0, F, $S_{n,m}$, p_n, k_n^t, k_n^e

Output: $Cost^{total}$

1: K is number of offloading devices

2: $Cost^{total} \leftarrow 0$

3: $\boldsymbol{for}\ n = 1, 2, \dots, N$

4: $\boldsymbol{for}\ m = 1, 2, \dots, M$

5: **if** $z_{n,m} = 0$

6: Calculate time $T_{n,m}^l$ according to eq. (4)

7: Calculate energy consumption $E_{n,m}^l$ according to eq. (5)

8: Calculate the cost $Cost_{n,m}^l$ according to eq. (6)

9: $Cost^{total} \leftarrow Cost^{total} + Cost_{n,m}^l$

10: **else**

11: Allocate the resource allocation $according\ to\ C_n^s \leftarrow \frac{F}{K}$

12: Calculate channel gain according to eq. (8), (9)

13: Calculate uplink rate according to eq. (7)

14: Calculate time $T_{n,m}^s$ according to eq. (12)

15: Calculate energy consumption $E_{n,m}^s$ according to eq. (13)

16: Calculate the cost $Cost^{s}_{n,m}$ according to eq. (14)

17: $Cost^{total} \leftarrow Cost^{total} + Cost^{s}_{n,m}$

18: **end**

19: **end**

20: **end**

18: **return** $Cost^{total}$

4.4 Optimal offloading strategy

Optimal offloading strategy, Algorithm 5, is the method, which used to find the best offloading decision to minimize the total cost. All possible offloading decisions for the number of devices with their tasks (2NM) in the system are first generated by the algorithm, then the total cost $Cost^{total}$ of all possible offloading decisions is determined. Finally, return the minimum total cost $Cost^{total}_{min}$. However, this algorithm will only operate for up to 7 devices with 3 tasks for each device, because it is a complex method, which takes a long time to generate all possible decisions and compute the optimal offloading, especially when number of users and tasks increases.

Algorithm 5: Optimal offloading strategy

Input: N, M
Output: $Cost^{total}_{min}$
$Cost^{total}_{min} \leftarrow \infty$
1: ***for*** $z \in \{0,1\}^{NM}$
2: Calculate $Cost^{total}$ *according to Algorithm* (4)
3: **if** $Cost^{total} < Cost^{total}_{min}$ *to* calculate the minimum cost
4: $Cost^{total}_{min} \leftarrow Cost^{total}$
5: end
6: end
7: **return** $Cost^{total}_{min}$

4.5 Reinforcement learning

In this section, we will show the main components of reinforcement learning, followed by a detailed demonstration of deep reinforcement learning method for generating the offloading decision. where the reinforcement learning is a category of the machine learning.as shown in Fig. 2, the important components of reinforcement learning are state, action, reward, environment, policy, and agent. where the agent takes the state from the environment. the action a(t) is selected from action space A, where the probability to choose action a(t) depend on the policy $\pi = P(a(t)|\mathcal{Z}(t))$. the selected action a_t is required to move from state $\mathcal{Z}(t)$ to next state $\mathcal{Z}(t+1)$. the objective of the reinforcement learning is increasing the cumulative rewards which calculated by: $R = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$, where γ is discount factor [36][37]. The discount factor is number between (0,1). The solving of the optimization problems is by The Q learning algorithm and Deep DQN algorithm. The main parameters are state, action, reward, and policy [30].

- State: the state Z(t) is the offloading decision Z= {z_1,1,z_1,2,…,z_(N,M)}, where State space Z is 1 X NM vector is defined as:
$$\mathcal{Z}(t) = \{z_{1,1}(t), z_{1,2}(t), \dots, z_{N,M}(t)\} \quad (17)$$

- Action :the action space A is 1 X NM vector, where the selecting action is required to move from state to next state, where a_tis the index selection from state length. The index selection l = 1, 2, 3, …, NM and the action is defined as:
$$a(t) = \{1,2,\dots,NM\} \quad (18)$$

Reward: the reward value $r(\mathcal{Z}(t), a(t))$ depend on the objective function, where the objective function of the optimization problem is minimizing the time and energy consumption to execute the tasks of all wireless devices. In our problem, the objective function is derived from the state $\mathcal{Z}$ (t) and eq. (15), where denoted by eq. (19):

- $$Cost^{total}_{\mathcal{Z}(t)}(t) = \sum_{n=1}^{N}\sum_{m=1}^{M}(1 - z_{n,m})\, Cost^{l}_{n,m}(t) + z_{n,m}\, Cost^{s}_{n,m}(t) \quad (19)$$

The reward value of state s(t) and action a(t) is calculated by the eq. (20):

$$r_{s(t),a(t),s(t+1)} = \begin{cases} 1 & cost_{\mathcal{Z}(t)}(t) > cost_{\mathcal{Z}(t+1)}(t+1) \\ -1 & cost_{\mathcal{Z}(t)}(t) \le cost_{\mathcal{Z}(t+1)}(t+1) \end{cases} \quad (20)$$

- Policy: the policy used to select the action in this work is ε greedy policy, ε ∈(0,1). At the beginning of learning we don't know the optimal action a for that the action is randomly chosen for ε probability to explore the environment this is called exploration. When the probability is $1 - \varepsilon$, The action $a = \arg\max_a Q(\mathcal{Z}, a)$ to choose the optimal action this is called exploitation.

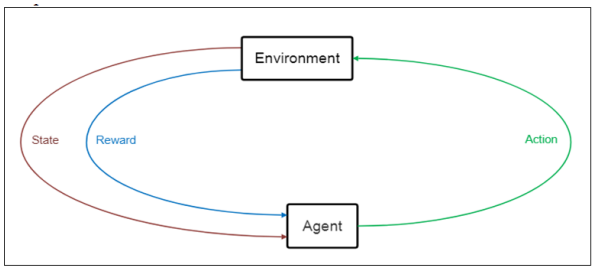

Fig. 2. Reinforcement Learning architecture

There are different models of reinforcement learning. In this study Q learning model are used. DQN model of deep reinforcement learning are used also to achieve the objective function. Q learning and DQN will be explained below.

4.5.1 Q learning

Q learning algorithm is a free model of reinforcement learning. This algorithm can solve our problem by using Q table [29][38] where the states $\mathcal{Z}$ are represented in rows of the table and the actions A is represented in the columns. first, initialize Q value in the table. At each step, the action is chosen by using ε greedy policy. The action is performed to move from state to next state and to calculate the reward according to eq. (19), then the Q value is update according to eq (20).

$$Q(\mathcal{Z},a) = Q(\mathcal{Z},a) + \alpha[r(\mathcal{Z},a) + \gamma \max Q(\mathcal{Z}',a') - Q(\mathcal{Z},a)] \quad (21)$$

Where the $\mathrm{Q}(\mathcal{Z},\mathrm{a})$is the old value of $\mathrm{Q}(\mathcal{Z},\mathrm{a})$, $\max \mathrm{Q}(\mathcal{Z}',\mathrm{a}')$ is maximum expected future reward value of Q in the next state respectively, $\mathrm{r}(\mathcal{Z},\mathrm{a})$ is the reward, the left $\mathrm{Q}(\mathcal{Z},\mathrm{a})$ is the update of $\mathrm{Q}(\mathcal{Z},\mathrm{a})$, value α is the learning rate, and γ is the discount factor. where $\alpha \in (0,1)$and $\gamma \in (0,1)$.

Algorithm 6: Q learning [29]

Input: N, M, α, γ

Output: $Cost_{min}^{total}$

1: Set $\mathcal{Z}$ to full local
1: Initialize $Q(\mathcal{Z},a)$
2: ***for*** $ep = 1, episodes\ do$
3: Choose initial $\mathcal{Z}$
4: With probability ε select a random action $a(t)$
5: otherwise select $a(t) = max_a Q(a)$
6: ***for*** $numberofsteps\ do$
7: Execute $a(t)$ and calculate $cost_{\mathcal{Z}(t)}(t)$ according to eq. (19)
8: Calculate $r(t)$ according to eq. (20)
9: if $\mathcal{Z}(t+1)$ is terminal state
10: break
11: **end**
12: $Q(\mathcal{Z}(t),a(t)) \leftarrow Q(\mathcal{Z}(t),a(t)) + \alpha[r(t) + \gamma \max Q(\mathcal{Z}(t+1),a) - Q(\mathcal{Z}(t),a(t))]$
13: $\mathcal{Z}(t+1) \leftarrow \mathcal{Z}(t)$
14: **end**
15: **end**
16: $Cost_{min}^{total} \leftarrow total\ cost(s): s\ is\ the\ terminal$
17: return $Cost_{min}^{total}$

4.5.2 Deep Q Network

DQN is a branch of Deep Reinforcement Learning algorithm, where DQN used the DNN is used to generate and save Q values.as shown in Fig. 3, DQN [39][40] is consist of two DNNs, where the first DNN is the evaluation network and the second DNN is the target network as shown in fig. The parameter values of evaluation network and target network are $\theta, \acute{\theta}$ respectively. The evaluation network and the target network generate the outputs $\mathrm{Q}^{\mathrm{pre}}(\mathcal{Z},\mathrm{a})$ and $\mathrm{Q}^{\mathrm{tar}}(\mathcal{Z},\mathrm{a})$ respectively. The inputs of evaluation network and the target network are the state $\mathcal{Z}(\mathrm{t})$ and the next state $\mathcal{Z}(\mathrm{t}+1)$ respectively. In the beginning choose initial state. The action is selected by using ε greedy to move to next state and calculate the reward according to equation (18). Save state, next state, action, reward, and done in replay memory B. Take a sample random mini batch from memory B then Calculate $\mathrm{Q}^{\mathrm{lab}}$:

$$Q^{lab} = \begin{cases} r(t) & for\ terminal\ state\ \mathcal{Z}(t+1) \\ r(t) + \gamma\ \mathrm{máx}_a\ Q^{tar}(\mathcal{Z}(t+1), \acute{a}; \acute{\theta}) & for\ non\ terminal\ state\ \mathcal{Z}(t+1) \end{cases} \quad (22)$$

Use a gradient descent algorithm to optimize the parameter values θ of DNN to minimize the loss, where the loss function is calculated by:

$$\text{Loss} = (Q^{lab} - Q^{pre}(\mathcal{Z}(t), a(t); \theta))^2 \quad (23)$$

Finally, after C steps, update $\acute{\theta} = \theta$. The DQN algorithm is showed in Algorithm 7 in details.

Algorithm 7: Deep Q Network [30]

Input: State $\mathcal{Z}(t)$ is offloading decision

Output: Optimal offloading decision

1: Generate target and evaluation Q network with a random parameter θ , θ'
2: Generate the empty replay memory with size m
3: ***for*** $ep = 1, episodes\ do$
4: Choose initial state
5: ***for*** $t = 1,2,3,\ldots, T\ do$
6: With probability ε select a random action $a(t)$
7: otherwise select $a(t) = arg\ max_a Q^{pre}(\mathcal{Z}(\mathrm{t}), a(t); \theta)$
8: Execute $a(t)$ and calculate $cost_{\mathcal{Z}(t)}(t)$ according to eq. (19)
9: calculate $r(t)$ according to eq. (20)
10: **if** $\mathcal{Z}(t+1)$ is terminal state
11: done is true

12: **end**

13: Save transition ($\mathcal{Z}(t+1)$, $a(t)$, $r(t)$, $\mathcal{Z}(t+1)$,done) in memory

18: Sample random minibatch of transitions from memory

14: Calculate Q^{lab} according to eq. (22).

15: Calculate loss according to eq. (23).

16: Make $\acute{\theta} = \theta$ after each C steps

17: $\mathcal{Z}(t+1) = \mathcal{Z}(t)$

18: **if** done is true

19: break

20: **end**

21: **end**

22: **end**

4.6 Distributed Deep Neural Network strategy

A distributed deep neural network (DNN)-based task offloading algorithm, Algorithm 8, [26][41]-[43] is performed to solve the problem of eq. (16), where it consists of multiple deep neural network (DNN) as shown in Fig. 4. Y DNNs have the same neural network as number of layers, nodes, and activation function, where Y is number of DNNs. In addition, the memory experience is shared between all DNNs. At each episode t, Y DNNs take each input state s_t to produce different binary offloading $z_{y,t}$ according to $f_{\theta_{y,t}}: S_t \rightarrow z_{y,t}$, where $f_{\theta_{y,t}}$ is a parametrized function, $\theta_{y,t}$ is the parameter of y^{th} DNN, $y \in \mathcal{Y} = \{1, 2, \ldots, Y\}$ is the index of DNN. Then the offloading decision with the lowest cost is selected as the output according to eq. (24) then save it in the memory experience to use it in the training of distributed DDN algorithm.

$$z_t^* = \underset{y \in \mathcal{Y}}{\boldsymbol{arg\,min}} \; \boldsymbol{cost}^*(\boldsymbol{S}_t, \boldsymbol{z}_{y,t}) \quad (24)$$

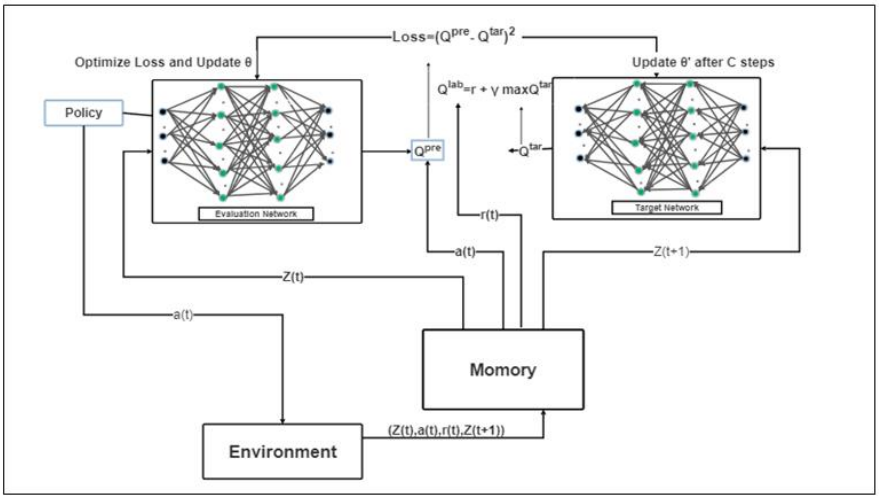

Fig. 3. DQN with the environment MEC [30]

Algorithm 8: Distributed DNN-Based Task Offloading [26]

Input: different data size S_t

Output: The best offloading decision z_t^*

1: Assign random parameters $\theta_{y,t}$ to all DNNs $where\ y \in \mathcal{Y}$

2: Generate an empty memory of size p

3: $\boldsymbol{for}\ \ t = 1,2,3,\ldots,T$

4: Enter the same S_t to each DDNs

5: Create the output offloading decision from all DNNs

$\{z_{y,t}\} = f_{\theta_{y,t}}(S_t)$

6: Select the offloading decision according to eq. (24)

7: Save (S_t, z_t^*) in the memory

8: Select a random mini batch from the memory

9: train DNN and update $\theta_{y,t}$

10: **end**

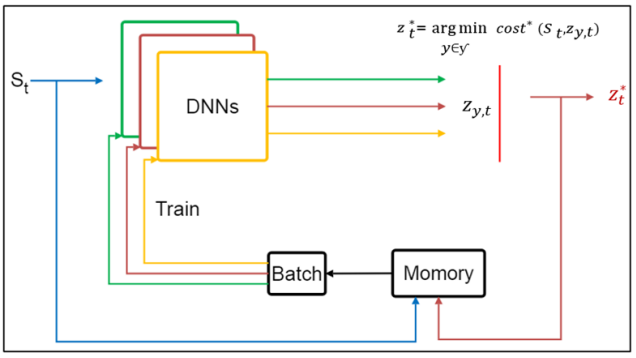

Fig. 4. Distributed DNN-based task offloading architecture

Neural Network

The neural networks are used in deep Q network and a distributed deep learning-based task offloading have the same number of inputs and outputs, and layers, and the same activation function, where the neural networks is consisting of four fully connected linear layers: Input and output layer have NM nodes, two hidden layers where the first hidden layer has 120 nodes, and the second hidden layer has 80 nodes. The activation function of the hidden layers and the output layer are Relu and Sigmoid, respectively [28]-[30], and Adam algorithm is the optimizer. But they have different loss function, where the loss function in DQN is a mean square error [30] and a cross entropy loss function in Distributed DNN [41][42].

Table 3 : Simulation Parameters

MEC Parameters [30]	
Number of wireless devices N	4
Number of tasks M	3
Bandwidth B	20 MHz
Transmitted power p_i	100 mw
Background noise N_0	- 100 dBm
Size of data $S_{i,j}$	(0, 10) MB
Number of CPU cycles per bit C	1000 Cycles/bit
CPU frequency of each device C_i^l	0.6 GHz
Energy consumption per CPU cycle β_i	(0, 20 x 1011) J/Cycle
CPU frequency of edge server F_c	100 GHz
Q learning Parameters	
Number of episodes	4000
Number of steps	8
Learning rate α	0.01
Discount factor γ	0.99
DQN Parameters	
Number of episodes	10000
Memory size	1024
Batch size	32
Learning rate α	0.1
Discount factor γ	0.99
Greedy value ε	1
Minimum greedy value	0.1
Decay greedy per episode	0.995
Number of steps to update target network	100000
Distributed DNN Parameters	
Memory size	512
Batch size	128
Number of DNN	5
Learning rate α	0.0001

5. RESULTS

In this section, we study the result of each strategy which explained in section 4 to produce the offloading decision. Our results were simulated at Google Colaboratory with Intel(R) Xeon(R) CPU @ 2.30GHz Processor and 12.0 GB available RAM. Python is the simulator of our work, where PyTorch library are used to establish the neural network. The parameters of the simulation in Tables 3.

5.1 Total cost with different number of users

In this section, we compare between different strategies to minimize the total cost. First, we calculate the total cost with different number of users. From Fig. 5, it is noted that the full offloading strategy is better than full local, the optimal offloading is complex when number of users increase, where the number of iterations to find the optimal offloading decision equal 2NM, when number of devices is 7 and number of tasks 3 the iteration is 221=2097152. The Deep learning became more able to deal with more complex problem. As shown in figure, Q learning, DQN, and a distributed DNN algorithms achieve the optimal offloading. However, Q learning, when number of devices increase the Q table size increases, so it can't update the Q table. the DQN and distributed DNN algorithms solved the problem of increasing number of users. The results of simulation (4 devices with 3 tasks for each device) show that the total cost in terms of time and energy consumption in Q learning, DQN and, Distributed

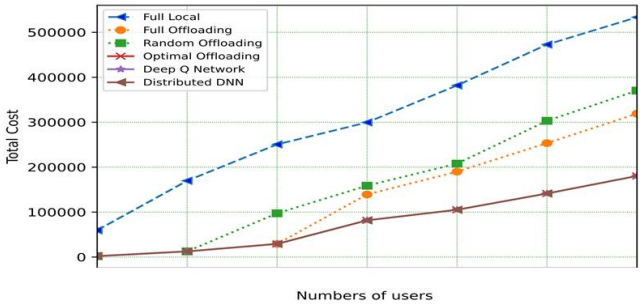

Fig. 5: Total cost when using different number of devices.

DNN algorithms is equal to the optimal offloading strategy, furthermore they reduce the total cost up to 63.7% when compared to full local strategy, also up to 21.8% when compared to full edge strategy. However, the DQN algorithm would need to train for a longer time than a distributed DNN when number of user increases.

In this subsection, we explain the parameters which effect on the convergence performance of a distributed DNN and DQN algorithms. There are different values of each parameter which are applied then we select the best value for our simulation

5.1.1 Distributed DNN Parameters

In this subsection, the effects of various parameters on the total cost will be discussed, which are number of DNN, Memory size, Batch size, Learning rate. The effect of increasing the number of DNNs on convergence performance is shown in Fig.6 (a) It is observed that the distributed DNN algorithm's convergence performance improves as the number of DNNs increases. Furthermore, when we use a single DDN, it can't learn anything from the data that comes out from itself. So, the number of DDNs is at least 2 DNNs.

In Fig.6(b), the total cost is plotted when memory size of a distributed DNN differs from 512 to 2048. The figure shows that the smaller memory size can achieve faster convergence. So, the memory size is set as 512,

where it is the proper value for the better convergence performance in a distributed DNN simulation.

In Fig.6 (c), the simulation results are shown with batch size varying from 32 to 128, where batch size is the number of samples which taken randomly from the memory size. The figure show that when the batch size is small, the converges is fastest, but with the low performance. It because small batch can only store small experience. In addition, it is true that the training takes longer time with a larger batch. Thus, from numerical results are showed in figure, we select batch size as 128 in distributed DNN simulation.

The effect of different learning rate ranging from 0.1 to 0.0001 on the convergence of distributed DNN algorithm is shown in Fig.6 (d) The convergence speed is faster when the learning rate increases. However, when learning rate increases to 0.1 or .01, the converge is faster but the local optimum occurs and lower performance. So, the learning rate is set to the proper value according to the situations. In a distributed DNN simulation, the proper value of learning rate is 0.0001.

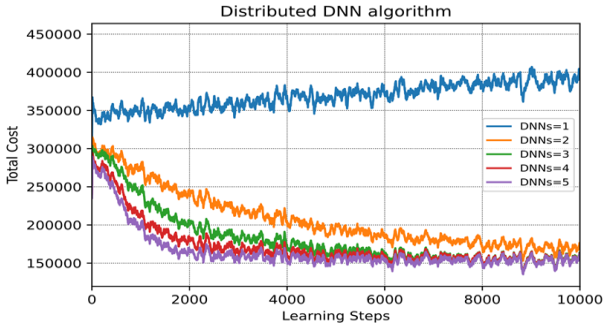

Fig.6 (a) Impacts of different numbers of DNNs

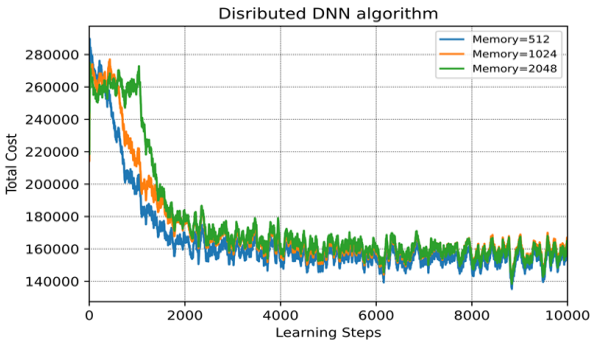

Fig.6 (b) Impacts of different Memory Size

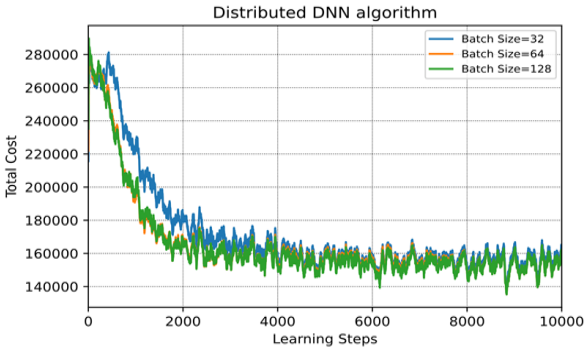

Fig.6 (c) Impacts of different Batch Size

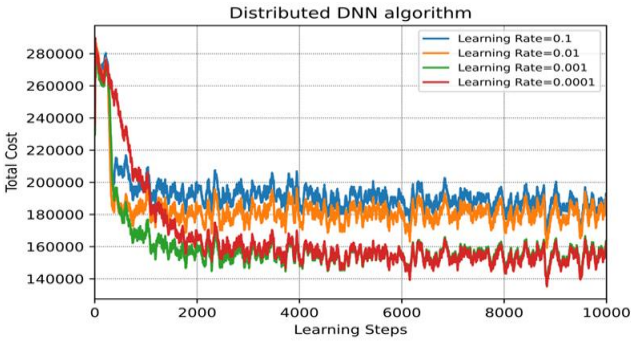

Fig.6 (d) Impacts of different Learning Rate

5.2 Deep Q Network Parameters

We also adjusted the parameters of Deep Q Network algorithm, which effect on the cumulative reward. Fig.7 (a) shows the impact of learning rates varying from 0.1 to 0.001 on the cumulative reward. From the figure, we can observe cumulative reward remained stable around 6 at the end. Furthermore, we discover that when learning rate is 0.01, the curve converges about 6000 learning steps. In addition, when learning rate is 0.001, the convergence process will reach a local optimal solution. So, learning rate is set to 0.1.

The impacts of various memory size ranging from 512 to 2048 on convergence performance are shown in Fig.7 (b). from the figure, when memory size is 512, the convergence performance results in a local optimal solution. Hence, the memory size set to 1024.

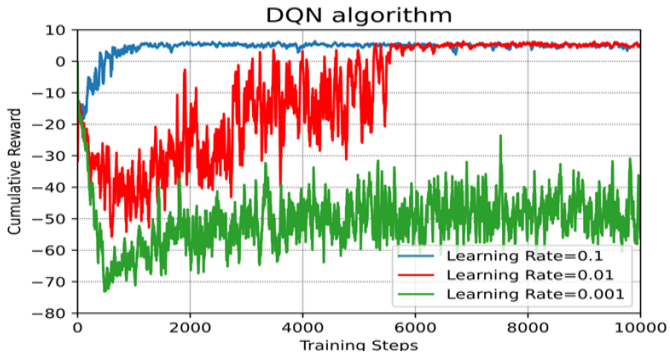

Fig.7 (a) Impacts of different Learning Rate

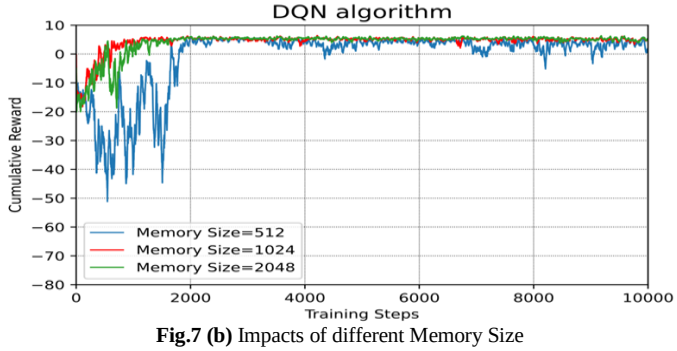

Fig.7 (b) Impacts of different Memory Size

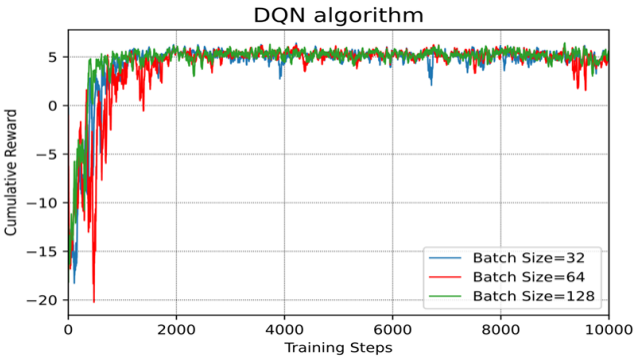

Fig.7 (c) Impacts of different Batch Size

Fig.7 (c) depicts how batch size effects on the cumulative reward and convergence performance. Although, the convergence happens more quickly as the batch size increases, it can be seen from the figure that the performance is higher when the batch size is 128 compared to 32 or 64. So, the batch size is set to 128 in DQN simulation.

In Fig. 8, it shows that total cost by using Distributed DNN and Deep Q Network algorithm. From figure, it is shown that after 6000 episodes became DQN and a distributed DNN have the same value of the total cost equal after 6000 learning steps. However, a distributed DNN generates the offloading decision for 4 devices with 3 tasks for each device (in 4 milliseconds) faster than DQN algorithm (in 8 milliseconds).

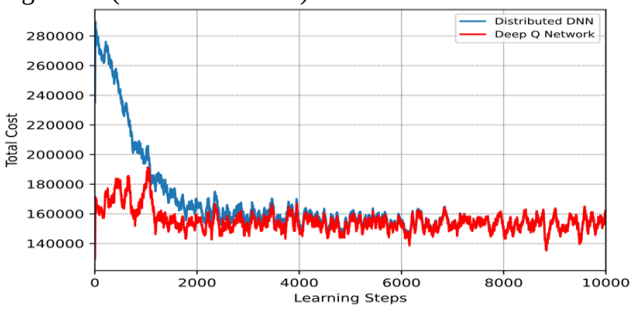

Fig.8 The convergence performance of Distributed DNN and DQN algorithm

6. CONCLUSION AND FUTURE WORK

This paper proposed a resource allocation and task offloading model for multiple-device, multiple-task MEC system. Furthermore, the model is formulated as a problem with the objective of decreasing the total cost in terms of time of computation and energy consumption. The proposed problem is solved by applying the six strategies, which they are compared with the optimal offloading strategy. Although, optimal strategy is an important guideline for the six strategies used, the optimal strategy does not scale as the number of devices increases. The simulation results show that DQN and a distributed DNN algorithms can outperform full offloading and full local strategies by up to 21.8%, and 63.7% of the total cost. Furthermore, the distributed DNN generates the optimal offloading decision faster than DQN in 4 milliseconds, but DQN in 8 milliseconds. So, it thinks that the distributed DNN algorithm can be further enhanced to improve real time offloading of MEC system.

In the future, we will investigate the task offloading problem in the MEC system with multiple edge servers, multiple devices, and multiple tasks. In addition, the study will focus on the mobility of the wireless devices, which enables the devices to move between the different Base Stations while the task is being offloading.

REFERENCES

[1] Fu, Zhangjie, et al. "Enabling personalized search over encrypted outsourced data with efficiency improvement." IEEE transactions on parallel and distributed systems 27.9 (2015): 2546-2559.

[2] Alghamdi, Ibrahim. Computation offloading in mobile edge computing: an optimal stopping theory approach. Diss. University of Glasgow, 2021.

[3] Yadav, Rahul, et al. "Adaptive energy-aware algorithms for minimizing energy consumption and SLA violation in cloud computing." IEEE Access 6 (2018): 55923-55936.

[4] Miettinen, Antti P., and Jukka K. Nurminen. "Energy efficiency of mobile clients in cloud computing." 2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 10). 2010.

[5] Wan, Shaohua, et al. "Efficient computation offloading for Internet of Vehicles in edge computing-assisted 5G networks." The Journal of Supercomputing 76.4 (2020): 2518-2547.

[6]

[7] Chen, Xu. "Decentralized computation offloading game for mobile cloud computing." IEEE Transactions on Parallel and Distributed Systems 26.4 (2014): 974-983.

[8] Fernando, Niroshinie, Seng W. Loke, and Wenny Rahayu. "Mobile cloud computing: A survey." Future generation computer systems 29.1 (2013): 84-106.

[9] Fernando, Niroshinie, Seng W. Loke, and Wenny Rahayu. "Mobile cloud computing: A survey." Future generation computer systems 29.1 (2013): 84-106.

[10] Zhang, Ke, et al. "Energy-efficient offloading for mobile edge computing in 5G heterogeneous networks." IEEE access 4 (2016): 5896-5907.

[11] Abbas, Nasir, et al. "Mobile edge computing: A survey." IEEE Internet of Things Journal 5.1 (2017): 450-465.

[12] Pham, Quoc-Viet, et al. "A survey of multi-access edge computing in 5G and beyond: Fundamentals, technology integration, and state-of-the-art." IEEE Access 8 (2020): 116974-117017.

[13] Cao, Bin, et al. "Intelligent offloading in multi-access edge computing: A state-of-the-art review and framework." IEEE Communications Magazine 57.3 (2019): 56-62.

[14] Omland, Sondre Eide. Deep Reinforcement Learning for Computation Offloading in Mobile Edge Computing. MS thesis. The University of Bergen, 2022.

[15] Mao, Yuyi, et al. "A survey on mobile edge computing: The communication perspective." IEEE communications surveys & tutorials 19.4 (2017): 2322-2358.

[16] Mach, Pavel, and Zdenek Becvar. "Mobile edge computing: A survey on architecture and computation offloading." IEEE communications surveys & tutorials 19.3 (2017): 1628-1656.

[17] Sadatdiynov, Kuanishbay, et al. "A review of optimization methods for computation offloading in edge computing networks." Digital Communications and Networks (2022).

[18] Zhang, Jing, et al. "Joint computation offloading and resource allocation optimization in heterogeneous networks with mobile edge computing." IEEE Access 6 (2018): 19324-19337.

[19] Mao, Yuyi, Jun Zhang, and Khaled B. Letaief. "Dynamic computation offloading for mobile-edge computing with energy harvesting devices." IEEE Journal on Selected Areas in Communications 34.12 (2016): 3590-3605.

[20] Salmani, Mahsa, and Timothy N. Davidson. "Uplink resource allocation for multiple access computational offloading." Signal Processing 168 (2020): 107322.

[21] Elgendy, Ibrahim A., et al. "Resource allocation and computation offloading with data security for mobile edge computing." Future Generation Computer Systems 100 (2019): 531-541.

[22] Elgendy, Ibrahim A., et al. "Efficient and secure multi-user multi-task computation offloading for mobile-edge computing in mobile IoT networks." IEEE Transactions on Network and Service Management 17.4 (2020): 2410-2422.

[23] Wan, Zhilan, et al. "Joint computation offloading and resource allocation for NOMA-based multi-access mobile edge computing systems." Computer Networks 196 (2021): 108256.

[24] Pławiak, Paweł, et al. "DGHNL: A new deep genetic hierarchical network of learners for prediction of credit scoring." Information Sciences 516 (2020): 401-418.

[25] Yang, Xiao, et al. "Communication-constrained mobile edge computing systems for wireless virtual reality: Scheduling and tradeoff." IEEE Access 6 (2018): 16665-16677.

[26] Ali, Zaiwar, et al. "Smart computational offloading for mobile edge computing in next-generation Internet of Things networks." Computer Networks (2021): 108356.

[27] Khayyat, Mashael, et al. "Advanced deep learning-based computational offloading for multilevel vehicular edge-cloud computing networks." IEEE Access 8 (2020): 137052-137062.

[28] Li, Yunzhao, et al. "Distributed edge computing offloading algorithm based on deep reinforcement learning." IEEE Access 8 (2020): 85204-85215.

[29] Huang, Liang, Suzhi Bi, and Ying-Jun Angela Zhang. "Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks." IEEE Transactions on Mobile Computing 19.11 (2019): 2581-2593

[30] Elgendy, Ibrahim A., et al. "Joint computation offloading and task caching for multi-user and multi-task MEC systems: reinforcement learning-based algorithms." Wireless Networks 27.3 (2021): 2023-2038.

[31] Elgendy, Ibrahim A., et al. "Advanced deep learning for resource allocation and security aware data offloading in industrial mobile edge computing." Big Data (2021).

[32] Chen, Liming, et al. "Intelligent Mobile Edge Computing Networks for Internet of Things." IEEE Access (2021).

[33] Yang, Shicheng, Gongwei Lee, and Liang Huang. "Deep Learning-Based Dynamic Computation Task Offloading for Mobile Edge Computing Networks." Sensors 22.11 (2022): 4088.

[34] Chen, Wen, Yuhu Chen, and Jiawei Liu. "Service migration for mobile edge computing based on partially observable Markov decision processes." Computers and Electrical Engineering 106 (2023): 108552.

[35] Liao, Linbo, et al. "Online computation offloading with double reinforcement learning algorithm in mobile edge computing." Journal of Parallel and Distributed Computing 171 (2023): 28-39.

[36] Bi, Suzhi, et al. "Lyapunov-guided deep reinforcement learning for stable online computation offloading in mobile-edge computing networks." arXiv preprint arXiv:2010.01370 (2020).

[37] Chen, Wen, et al. "A multi-user service migration scheme based on deep reinforcement learning and SDN in mobile edge computing." Physical Communication (2021): 101397.

[38] Zhu, Hao, et al. "Deep reinforcement learning for mobile edge caching: Review, new features, and open issues." IEEE Network 32.6 (2018): 50-57.

[39] Jiang, Kai, et al. "A q-learning based method for energy-efficient computation offloading in mobile edge computing." 2020 29th International Conference on Computer Communications and Networks (ICCCN). IEEE, 2020.

[40] Chen, Juan, and Zongling Wu. "Dynamic Computation offloading with Energy Harvesting Devices: A Graph-based Deep Reinforcement Learning Approach." IEEE Communications Letters (2021).

[41] Huang, Liang, et al. "Deep reinforcement learning-based task offloading and resource allocation for mobile edge computing." International Conference on Machine Learning and Intelligent Communications. Springer, Cham, 2018.

[42] Huang, Liang, et al. "Distributed deep learning-based offloading for mobile edge computing networks." Mobile networks and applications (2018): 1-8.

[43] Huang, Liang, et al. "Multi-server multi-user multi-task computation offloading for mobile edge computing networks." Sensors 19.6 (2019): 1446.

[44] Mukherjee, Mithun, et al. "Distributed deep learning-based task offloading for UAV-enabled mobile edge computing." IEEE INFOCOM 2020-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS). IEEE, 2020.

[45] Wen, Yonggang, Weiwen Zhang, and Haiyun Luo. "Energy-optimal mobile application execution: Taming resource-poor mobile devices with cloud clones." 2012 proceedings IEEE Infocom. IEEE, 2012.