Dynamic Resource Allocation and Streaming in Mobile Edges: A Deep Reinforcement Learning Approach

Daud Khan[1] and Zeeshan Pervaiz[2(✉)]

[1] School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai, China
Daud95@sjtu.edu.cn
[2] School of Computer Science, Bahria University, Islamabad, Pakistan
ZeeshanPervaiz332@gmail.com

Abstract. Real-Time wireless communication devices with restricted assets face more extreme limit limitations than at any other time expansion of various sophisticated and calculation severe, versatile applications. In this paper, we explore the issue of resource allocation and also use real-time devices in mobile edge networks, efficient streaming with Double Deep Q Reinforcement Learning. The ideal arrangement considering the elements the system is strict about accomplishing. We aim to develop a smart agent to improve the allocation of resources in the decision-making process. We present a new combination of double Q-learning and DQN dueling algorithms and design suggested a solution to this issue based on the Double Deep Reinforcement Learning. We implement Double Deep Q Reinforcement Learning that can take into consideration a long-term task and learn from experience. The current proposal also measures the time-varying tasks of MEC servers and discusses the strategy of transferring tasks from one to another MEC server, better optimizing the value of the task by reducing unnecessary waiting times for queue. Results from the simulation show that our proposed approach significantly decreases system costs relative to the other parameters.

Keywords: Mobile Edge Computing · Resource allocation · Deep Reinforcement Learning · Real-time · Double Deep Q-network

1 Introduction

5th generation wireless systems (5-G) is one of the notable modern-day innovations. This is the case that the 5-G era was considered on advancement in networking technology, several new communication computing and models are implemented, such as hyper-dense networks, millimeter-wave communications, and adjacent device-to-device networks [1–3]. Overall techniques, Mobile Edge Computing (MEC), is one of the vital computer/data delivery technologies for

M. Guan and Z. Na (Eds.): MLICOM 2020, LNICST 342, pp. 170–183, 2021.
https://doi.org/10.1007/978-3-030-66785-6_19

improving the performance of cellular networks [4]. With the advantages of MEC wireless mobile users technologies, high-speed, efficient, and real-time wireless communication services can be enjoyed, including data rate enhancements and efficiency improvements. Gathering with MEC technologies, numerous networked fundamentals are also concerned, such as connected devices and cloud servers. The number of linked devices is also expeditiously rising the volume of data transmission to the MEC. This potentially introduces significant constraints on the system, such as data transfer speed limits, restrictions on storage space, battery latency, and terminal device usage energy. Because of these drawbacks and challenges, it is more efficient to use a cloud computing network when dealing with big data than with local terminal computing. While MEC systems should be able to access or process data from connected devices in cases where data can not be managed in the cloud storage due to latency conditions and security reasons, so the trade can be observed between cloud computing services and MEC services. Overall, cloud servers have much more power and central processing advantages than MEC servers, while latency constraints and security constraints/regulations may have drawbacks.

Transmission from connected devices is delayed, which is much convenient for real-time processing; this allows data collection and real-time transmission using high-quality data for streaming in real-time or streaming videos. It is, therefore, necessary to develop a methodology that can handle the scheduling of data transmission. Notice that when a decision is taken to offload data from linked devices, cloud storage servers can handle this while MEC servers manage data when requested to offload it.

Besides, these drawbacks open the new era for the MEC field in different except, but they have some standard conditions. The MEC environment's randomness and complexity are not addressed. Instead of the binary offloading feature, these programs are usually shown as a mixed-integer program. To solve MIP problems, branch and bound Algorithms, and Dynamic Programming [5,6] have been implemented. The computational complexity, however, is incredibly high, particularly for large MEC networks. Local search heuristic [7] and convex relaxation method [8] are suggested to reduce the complexity of computations. As a consequence, several methods have emerged for applying machine learning to solve the resource allocation issue with cloud computing. In this paper, we present a new dynamic distribution of resources and wireless communication devices within a Real-time approach based on Double Deep Reinforcement Learning using double Q-learning, which can balance computation and system resources under various MEC conditions dramatically reduce total system costs and optimize system resources.

- In the Software Defined Network (SDN) controller, an intelligent model the Het-Nets to establish an adaptive resource allocation, a strategy that takes taking the offload tasks into account and considering the video streaming for multiple users.
- We are implementing the SDN controller software in our MEC architecture. The benefit of the SDN controller is that it logically centralizes the distributed

network infrastructure and improves the QoS. The joint optimal task, computing allocation, and streaming problem are developed and implemented to minimize system costs under constraints of limited and dynamic storage capacity and computing resources on devices and servers, as well as limitations on a device and hard deadline delay.
– We construct a double-deep Q-learning algorithm with a multi-time system to configure the variable function and measure resource allocation, as well as to determine the set of feasible connecting server and neighboring devices.
– We showed, Results of simulation to illustrate the performance of the proposed algorithms using the optimal variable configuration for a task, computing and streaming of the device. It studied the impact on system performance of device mobility, data size, back haul capabilities, and cloud resources.

2 Related Work

In [9] proposed a joint adaptive video streaming and an intelligent offloading approach based on deep Q-networks in mobile edge computing systems. Efficient streaming and offloading algorithms are essential to the utilization of video services in mobile edge networks. In [6], Reinforcement learning is studied in which agents learn by interaction with the environment and then make better experiential decisions. An agent acts, the reward is given on a scalar reward. The agent receives an overall accumulated reward in learning the technique. Adapting deep learning has helped RL to overcome issues in decision-making rapidly. Deep Reinforcement Learning-based algorithms are implemented in a broad variety of MEC issues. Q-learning is one of the well-known methods of reinforcement learning. The authors in [10,11] presented a dynamic resource allocation model based on the DRL in the MEC process. In [12], a Q-learning-based model was developed to reduce energy consumption during run time by dynamically adjusting the operating frequency. Approaches build the capacity to optimize dynamic power by learning optimal frequency management policies. But applying Q-learning has not always been able to get the desired results. Notably, In some stochastic and uncertain environments, the accuracy of selecting the optimal actions in Q-learning can not be ensured because a large number of overestimated Q-values are generated during the learning phase [6]. In [15] have shown that the overestimation of Q-learning is possible and has suggested a Double-Q learning method that illustrates its effectiveness in the Atari 2600 domain. In [13], proposed Double Deep Q-Network architecture for collaborative edge caching in mobile networks aims to reduce the long-term average content of mobile users. In [14], the problem of task scheduling in fog-based IoT applications has been studied to achieve an efficient time-and cost-saving approach within the resource and time constraint.

2.1 Motivation

Compared to current works, the primary benefit of this manuscript. We consider further steps to enhance the efficiency of resource allocation and offloading using

streaming devices for Real-Time Wireless communication. We make resource allocation and offload more effectively. We are proposing a new novel method considering the resource allocation task offload and streaming. Using Double Deep Reinforcement Learning using a recent new algorithm Double Q-Learning more specifically, When the task is offloaded the system allocates resources to terminal devices, including MEC servers, and ensures that the task is offloaded using the same servers. When the TD transfers the task to any other server. Moreover, If during the implementation of the MEC, specific existing computation resources were distributed, those resources could also be used to speed up computing. Hence, a systematic study to tackle the complexities of the MEC systems needs to be carried out to facilitate efficient resource management in a dynamic environment.

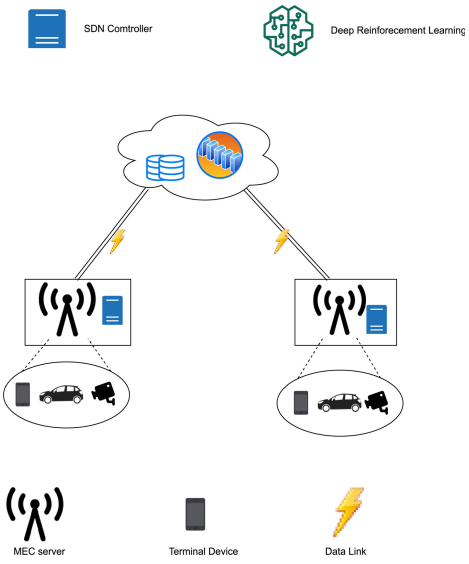

Fig. 1. MEC architecture.

2.2 Network Architecture

We design our Multi-server MEC server $M = 1, 2, 3, \ldots\ldots, M$ connected with our SDN and Terminal device $T = 1, 2, 3 \ldots\ldots, T$ are shown in Fig. 1. In which our SDN controller is considered with a MEC server to provide computation offloading service to the resources constrained users such as Augmented Reality (AR) and Virtual Reality (VR), drones, and smart phones. When delivering real-time video services to individual users such as video streaming and AR/VR, it is necessary to (i) preserve the high quality of video streaming under the tolerance latency constraint, and (ii) reduce the total execution time and total device

energy consumption. In General, all MEC servers can be the physical or virtual machine with specific computing capabilities provided by the network operator and can communicate with the devices through channels. Each user can choose to offload computation tasks to the MEC server for one of the nearby servers it can connect. The modeling of user computation tasks, task uploading transmissions, and offloading utility is shown here below.

2.3 Task and Computation Model

We assume υ a computational task for each TD. The η_υ task is described as the $(B_\upsilon, D_\upsilon, L_\upsilon)$ tuple, which can be executed on the local CPU of TD or the MEC server via computation offloading [16].

The B_υ length of the data required for the computation the D_υ length of each of the CPU cycles necessary to accomplish the task, and the L_υ task length is the maximum tolerable period of time, which ensures that the task does not exceed, L_υ length to satisfy the QoS requirement. We're showing $\alpha_\eta^\upsilon(t) = \alpha_{\eta(t)}^\upsilon = \alpha_\eta^\upsilon(t) \in 1, 2$ to the time slot t for TD υ of the offloading decision. We've got a $\alpha_\eta^\upsilon(t) = \alpha_\eta^\upsilon(t) = \alpha_\eta^\upsilon(t)$. If υ complete local computing task, $\alpha_\eta^\upsilon(t) = 1$. If υ offloads the function to the nearby MEC server, we will define the $A = [\alpha_\eta^1, \alpha_\eta^2, ..., \alpha_\eta^n]$ offload decision matrix. Requests generated by TD are allocated to MEC servers. A task that has been achieved but has not yet been completed will be sent to the first-in-first-out (FIFO) task queue q_m of MEC server m, with the accessibility of Q. Our SDN controller uses a DRL agent to track and interact with a region with time centrally, and an agent will receive the entry-level $x(t)$ and decide which action the policy will require at each point, a profit of each operation is a reward, and the agent seeks to optimize each Device State's cumulative reward.

If TD υ chooses to full the η_υ task by executing it's locally, T_η^l could be described as the local execution time of the η task, and we indicate f_υ^l (CPU cycles per second) as the TD υ computation efficiency. Therefore the local execution delay, T_η^l of the η_υ task is $T_\eta^l = \frac{D_\eta}{f_\upsilon^l}$ and the energy needed to complete the η_υ task is called $E_\eta^l = \varphi D_\eta$. φ represents the energy needed to complete the task per CPU cycle. The overall costs of η_υ local task execution can be expressed as $C_\eta^l = \delta^t T\eta^l + \delta^e E\eta^l$. Where δ^t and δ^e are the values, allow time and energy use to be prioritized and the value total is always equal to 1, where $\delta^t + \delta^e = 1$. If the TD υ wants to offload the computational task to represent edge server m at a time of t slot, the task input data must first be shifted to the servers edge. Data rate reachable is $r_t = \frac{W}{N} \log\left(1 + \frac{P_t \ h_t}{\frac{W}{N} \ N_0}\right)$. N is the number of offloading TDs, P_t is the transmission power for uploading data, h_t is channel gain of TD υ in the wireless channel, N_0 is the variance of the complex white gaussian channel noise. Transmission delay can be $T_\eta^t = \frac{B_\eta}{r_t}$ and the energy consumption is $E_\eta^t = P_\upsilon T_\eta^o = \frac{P_\upsilon B_\eta}{r_t}$. Where P_υ is the TD υ power unit needed transition of the data.

Completion time of a task can be estimated at a serving node as $T_\eta^o = \frac{L_\eta}{f_m^o}$. Where to complete the task, f_m^o can be defined as the allocated computational

resource (CPU cycles/second). Considering that the cumulative amount of overall resources will not exceed the total computing power on the MEC server. $\sum_{m=1}^{M} f_m^o \leq F$ the corresponding energy consumption is $E_\eta^o = P_\upsilon T_\eta^o$. The $W_{\upsilon,m}$ queuing time on the m MEC server is equal to the execution time for all tasks during the execution $\sum_{n=1}^{index(W_{\upsilon,m})} \emptyset\,(n)$ and $\emptyset\,(n)$ in the queue index stand for the execution time of the task n. Then the total execution time and energy consumption resulting from the discharge of tasks is η. $T_\eta = \frac{B_\eta}{r_t} + \frac{D_\eta}{f_m^o} + T_\eta^q$ and $E_\eta = \frac{P_\upsilon B_\eta}{r_t} + \frac{P_\upsilon^i L_\eta}{f_m^o}$. Where P_υ^i the stationary TD υ is power unit.

The cumulative cost of offloading computation can be determined as follows by comparing the execution delay and energy consumption.

$$C_\eta^l = \delta^t T_\eta + \delta^e E_\eta \tag{1}$$

and the total overall system cost of all users in the MEC offloading system is expressed as

$$C_{total} = \sum_{\upsilon=1}^{U} \sum_{\eta=1}^{W} (1 - \alpha_\eta^\upsilon) C_\eta^l + \alpha_\eta^\upsilon C_\eta^o \tag{2}$$

Where $\alpha_\eta^\upsilon \in 0, 1$ shows the TD υ offloading decision. If TD υ wants to perform the task locally then $\alpha_\eta^\upsilon = 0$ otherwise $\alpha_\eta^\upsilon = 1$. Also, the output size for edge servers is much smaller compared to the input data size. The task time from the server to the TDs is, therefore, being fairly overlooked.

2.4 Video Streaming

Video streaming is the increasing usage of today's bandwidth and absorbs about 70% of Internet traffic—Cisco's video steaming projection for 2020 amounts to about 82% of total Internet traffic [17]. In 2013, on average, there had been one 11-person surveillance camera in UK [18]. It can be seen that soon rising amounts of data will be sent to the Internet from devices. Given the higher demands for bandwidth, storage, and processing, managing these gigantic amounts of data will be difficult. Edge computing provides a feasible method for storing and filtering data before they are sent to the cloud.

Massive volumes of data can be stored, and required information or data can only be sent to the cloud for further processing or research storage, saving multi terabits per second of data. In addition to reducing bandwidth consumption from devices to the Internet, some of the applications, such as target monitoring, object assessment, surveillance, etc., are mission-critical real-time demand and aggregate contact from multiple sensors or locations, which is also 2017 2017Second International Fog and Mobile Edge Computing Conference (FMEC) 70 unfeasible in current cloud scenarios. Edge technologies can be used to integrate information from several sensors for real-time analysis and communication.

3 Problem Formulation

We formulate the quality of service and better support for Terminal Devices (TDs) specialized functionality; we aim to minimize the overall costs of all

offloaded tasks generated by Terminal Devices (TDs), distributed across the MEC region, within the limitation of maximum tolerable delay and computational power, the issue is formulated as follows.

$$min_{\mathcal{A},\mathcal{F},\mathcal{V}} \sum_{\upsilon=1}^{U} \sum_{\eta=1}^{W} (1-\alpha_{\beta}^{\upsilon}) C_{\eta}^{l} + \alpha_{\eta}^{\upsilon} C_{\eta}^{o}. \tag{3}$$

$$C1 : \alpha_{\eta}^{\upsilon}(t) \epsilon 0, 1, \forall \upsilon U \forall \eta . \epsilon W \tag{4}$$

$$C2 : (1-\alpha_{\beta}^{\upsilon}) T_{\eta}^{l} + \alpha_{\eta}^{\upsilon}(t) T_{\eta}^{q} \leq L_{\eta}. \tag{5}$$

$$C3 : \sum_{m=1}^{M} f_{m}^{o} \leq F, \forall \epsilon M. \tag{6}$$

The problem (3) is solved by detecting the optimal value $\mathcal{A}$ of which the decision vector is offloaded, the resource allocation vector $\mathcal{F}$, and the vector $\mathcal{V}$ Of the TD's streaming information the difference between the node and TD is described. $C1$ Is TD's decision to access or offload the task using local computing. $C2$ ensured the task's execution time could not surpass the maximum tolerable task delay. $C3$ reflects that the overall size of the computing resources allocated can not surpass the MEC server capacity as the number of TDs increases; the complexity of the problem can be increased. Rather than using the traditional methodology of optimization, we are implementing a new method of Deep Reinforcement Learning to evaluate the alternate solution.

We have introduced these problems in Markov Decision Process (MDP), time-space is subdivided into the slots of the same length is $\{t = 0, 1, 2, \ldots, \infty\}$. A be an action space and S be a State space. Then status of the MDP process for each t time is specified as $s_t \in s$, and $a_t \in A$ would be any action necessary in s_t state. Unless the a_t action is taken, which is available at s_t based on any policy, then the $r_t(s_t, a_t)$ and s_t reward is transformed to s_{t+1}, reward is a scalar quantity, and represents the efficacy of the executed action and vector for system operating would be represented by $(s_1, a_1, s_2, a_2, \ldots, s_n, a_n)$ sequence which could be described as the Markov Decision Process. The problems with the MDP can be defined as we can find an optimal control strategy, which in our case gives the highest cumulative reward, which is equal to the total system cost.

4 Dynamic Resource Allocation and Streaming in Mobile Edges: A Deep Reinforcement Learning Approach

The MDP is a policy, which maps for each State the best action in the MDP. This optimal policy can be found through a range of techniques, but we offer a Deep Reinforcement (DRL) approach due to the high complexity of our system. We claim that an SDN Controller integrates DRL Agents through edge computing.

The agent communicates with the system at all times and receives from the environment the corresponding input state.

$$x_t\{E_t, v_t, q_t, s_t\}. \tag{7}$$

where E_t = accumulate device costs at t, $v_t = [f^t_{m,1},\ f^t_{m,2},\ f^t_{m,N}]$ is a linear vector N consisting of available MEC server computing resources, which can be determined the following way. f^t_m At the time slot t, MEC server computing resources m are available. $s_t =$ video streaming is considering time t.

$$f^t_m = F - \sum_{n=1}^{N} \alpha^{\upsilon}_{\eta} f^o_n. \tag{8}$$

$q_t = [q^t_{m,1},\ q^t_{m,2},\ q^t_{m,N}]$ is a N linear vector consisting of the queue data for each MEC. MD υ is linked to one of the $m\ \in M$ serving MEC servers at each time step.

4.1 Action

Agent will decides to take action $a_t \in A$ from available actions at any time step t. The agent must decide which computing feature to offload, and which MEC server to allocate to the TD. Our action consists of the components which follow.

$$a_t = \{A_t, F_{t,}, \mathcal{T}_t\}. \tag{9}$$

Where $A_t = [\alpha^{\upsilon}_{\eta,1}(t),\ \alpha^{\upsilon}_{\eta,2}(t), \ldots,\ \alpha^{\upsilon}_{\eta,n}(t)]$. $\alpha^{\upsilon}_{\eta}(t) = \alpha^{\upsilon}_{\eta}(t)$ and $\alpha^{\upsilon}_{\eta}(t) \in \{0, 1\}$ indicates that terminal device decision υ at the time step t is offloaded. If TD υ decides to execute the task locally $\alpha^{\upsilon}_{\eta} = 0$ otherwise $\alpha^{\upsilon}_{\eta} = 1$. $F_t = \left[f^{\upsilon}_{m,1},\ f^{\upsilon}_{m,2}, \ldots,\ f^{\upsilon}_{m,n}\right]$. Where $f^t_m = f^t_m$ Indicates the allocated computational resources for the tasks offloaded by TD υ waiting in the task queue. Vector $\mathcal{T}_t = \left[\mathcal{T}^{\eta}_{m_1,m_2,1}(t), \mathcal{T}^{\eta}_{m_1,m_2,2}(t), \ldots,\ \mathcal{T}^{\eta}_{m_1,m_2,n}(t)\right]$. Where $\mathcal{T}^{\eta}_{m_1,m_2}(t)$ denotes the η task allocation action, at the time slot t waiting in the execution for task queue for $\mathcal{T}^{\eta}_{m_1,m_2}(t) = 0$, implies the task will be performed by the same server $\mathcal{T}^{\eta}_{m_1,m_2}(t) = 1$ which implies agent must decide to η transfer task from MEC server m_1 to m_2.

4.2 Reward

It includes evaluative feedback to a decision-making agent. The user adopt an action for each state and gets reward $r_t(s_t,\ a_t)$. In general, the reward feature is combined with the aim feature of the problem. Thus, the distribution of resources is aimed at minimizing cumulative device costs and meeting consumer demands, making our goal comes true. The reward feature is first designed to increase a negative cost during a choice to offload the task or perform the locally generated tasks. Instead, the DRL-based approach discovers a method for offloading with the highest overall reward. The best strategy in our framework will be the one

that will result in the lowest cost of execution and will reduce the overall cost of the system. The optimization reward is described by

$$max_{at}\mathbb{E}\left[\sum_{t=0}^{T} r_t(s_t, a_t)\right]. \tag{10}$$

The agent receives a r_t reward when engaging with the MEC system and agent seeks to obtain the highest possible R_t discount reward

$$R_t = \ r_t + \ \gamma R_{t+1}. \tag{11}$$

The discounted factor is γ, and the $0 \leq \gamma \leq 1$.

4.3 Deep Reinforcement Learning

Reinforcement Learning is a form of machine learning and works on enabling the agent to accurately determine the optimal behavior within a given context to maximize the cumulative return it predicts. Reinforcement Learning issues can be described as the MDP problem for optimal control decisions. Q-Learning, one of practical model-free. The agent has belonged in our architecture and obtains its currents state s_t. The agent then selects and operates a_t. Meanwhile, the environment undergoes a transition from s_t to a new s_{t+1} state and receives a $Q(s_t, a_t)$ reward The optimal state-value function $V(s)$ of Bellman Equation can be expressed as (12), where $s = s_t$ is the state at current decision epoch i, and the next state is $\acute{s} = s_{t+1}$ after take the action $a = a_t$.

$$V(s) = \max R(s, a) + \gamma \sum_{\acute{s}} P(\acute{s}|s, a).V(\acute{s}). \tag{12}$$

right hand side of (12) can be further expressed as

$$Q(s, a) = \ R(s, a) + \gamma \ \sum_{\acute{s}} P(\acute{s}|s, a).V(\acute{s}). \tag{13}$$

to obtained a optimal state value function $V(s) =$ would be expressed as

$$V(s) = \max Q(s, a). \tag{14}$$

Comparing (12), (13) can be expressed as

$$Q(s, a) = R(s, a) + \gamma \sum_{\acute{s}} P(\acute{s}|s, a).V(\acute{s}). \max \acute{a} Q(\acute{s}, \acute{a}). \tag{15}$$

Q-function formula can be written as

$$Q^{i+1}(s, a) = Q^i(s, a) + \alpha^i(R(s, a) + \gamma. \max \acute{a} Q^i(\acute{s}, \acute{a}) - \ Q^i(s, a) \tag{16}$$

Where $\alpha_i \in [0, 1]$ is the learning rate, and the s_t state turns to the s_{t+1} state when the agent choose action along with the corresponding $R(s_t, r_t)$ reward

based on (16), the Q table is used to store the Q value for each state-action pair when the dimensions of the state, as well as action space, are not high in the Q-Learning algorithm. Q-Learning process of learning is becoming relatively slow when the scenarios are with vast network states and action spaces. Deep learning techniques are then used as a possible method to approximate the value function (Fig. 2).

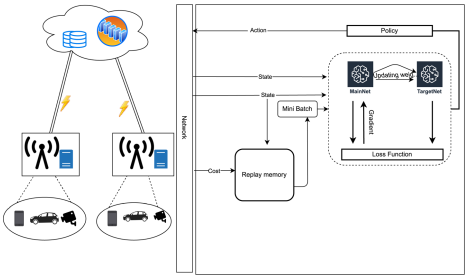

Fig. 2. Training process of double deep Q-network.

4.4 Double Deep Q-Network Method

Deep Q-Learning helps solve complicated and large-dimensional Reinforcement Learning problems and the multi-layer perceptron (MLP) structure implemented in it and can be used to estimate the optimal state-action of Q-functions. In our prototype, we use a full version of DQN to train our DRL agents, namely Double DQN [15]. The Q-function could be estimated by updating the variable θ of MLP to the optimal Q value as follows.

$$Q(s_t, a_t) = Q(s_t, a_t; \theta). \tag{17}$$

In fact, about the Double DQN characteristic, every DRL agent has two Q networks. $Q(s_t, a_t; \theta)$ and $\hat{Q}(s_t, a_t; \theta')$, with the Q network to select action, and the $\hat{Q}$ network to determine action. Remember those weight parameters of networkθ' $\hat{Q}$ are updated periodically by counterpart θ network Q. Algorithm-1 utilize Z of a N memory replay capability to save $Q(s_t, a_t, r_t, s_{t+1})$ transformation for each episode. Then sample a Z mini-batch transformation to train the Deep Neural Network (DNN) to a reduced loss feature; the estimated values are modified in each episode that follows.

A resource allocation framework based on Double Deep Q Network (DDQN) for the TDs within a MEC framework. Our DDQN uses a DNN with the values θ to estimate the Q-function. The agent will select an action that raises the Q-value at the state s_t and receives the reward r_t after the action is applied, then transitions to s_{t+1}.

Algorithm 1. Double Deep Q-Network (DDQN)

Initialize replay memory Z to Capacity N;
Initialize main Q network with random weightθ;
Initialize main Q' network with θ'=θ;
for $e = 1toE$ **do**
  Initialize state $s_1 = x_1$ and pre processed sequence $\phi_1 = \phi(s_1)$;
  for $i = 1toI$ **do**
    with probability ϵ choose random action a_i;
    Otherwise set $a_i = s_1, argmax_a Q(s_1; a_t; \theta)$;
    Execute the action a_i, observe reward r_1 and state $x_{(i+1)}$;
    set $s_{(i+1)}$=$s_i, a_i, x_{(i+1)}$;
    save transition $(s_i; a_i; r_i; s_{(i+1)})$ in Z;
    sample random the mini-batch of transitions $(s_i; a_i; r_i; s_{(i+1)})$ from Z;
    if (e finishes at k + 1) **then**
      assign $y_k = r_k$;
    else if then
      assign $y_k = r_k + (\gamma Q(s)_1, (argmax)_a Q(s_1; a_t; \theta))$.
    end if
    Compute gradient descent on $(y_k - Q(s_1; a_t; \theta))^2$
    Update the main Q network parameters θ;
    Update the main Q' network parameters θ';
  end for
end for

$$L(\theta_i) = E_{(s,a,R(s,a),\acute{s}) \in minibatch}\left[\left(R(s,a)+ [\gamma.\hat{Q}(\acute{s}\ arg\max \acute{a}Q(\acute{s},\acute{a};\theta) : \acute{\theta}) - Q(s,a;\theta)\right)^2\right] \tag{18}$$

In (18), the gradient guiding update. Therefore, Stochastic Gradient Descent (SGD) is conducted before Q networks converge to approximate optimum Q-function state-action. The Double DQN training algorithm is in Algorithm 1.

5 Simulation

We have performed multiple experiments to test our proposed system. It includes several edge servers and various TDs scattered around the zone. We take into consideration a scenario. The MEC server can be measured from 4 GHz second to 6 GHz s, and each MD has a CPU frequency of 1 GHz. We assume that the capacity of the transmission is 500 mW, and the idle power is 100 mW [19]. Data size B_η (in Kbits) for A task of [300, 500] is produced by the random means of W and the required computer resources are between [700; 1100] D_η (in Megacycles).

We compare our proposed DDQN with the other techniques to measure the efficacy of our proposed strategies Full Local, Full offload, greedy method [20]. All TDs which perform tasks locally can be defined as full local. Both TDs must

load their tasks entirely off to MEC servers rather than run them locally. Once MEC follows the greedy method and is unable to fulfill the application, the function is carried out locally to accomplish the minimum cost. The MEC server across all TDs will spread equal computing resources.

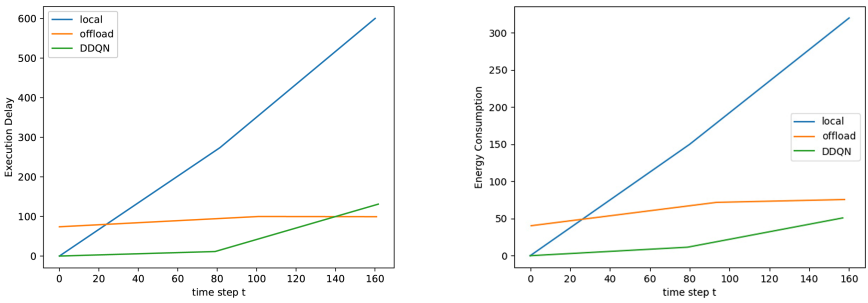

Fig. 3. Execution delay and energy consumption.

As shown in Fig. 3, the simulation performed with the results of full local computing (no offloading) and edge computing; we evaluate the effects of DDQN approach tasks, If tasks are not completed in time step t due to efficiency limit, all tasks are calculated as the tasks to be performed out in the next step. Hence, local computing increases the execution delay values and the energy consumption factor. When there are tasks that are not completed due to the ability limit at the time step t, all tasks are determined as the tasks to be taken out in the next stage. Local computing thus increases the execution delay values and the energy consumption factor. Besides, the time to perform tasks is the longest from time step 80, and it decreases the energy of TD as maximum.

The proposed DDQN approach has significantly better efficiency than TD's local computing. The achievement of the proposed DDQN approach is the lowest in terms of execution delay until time step 140. Indeed, when the time phase moves by and the task size increases, the outcomes of the recommended DDQN method are more significant than the maximum offload performance. The recommended DDQN-based approach also has the least values on the energy consumption side. These results reveal the DDQN algorithm proposed is the most relevant to real-time wireless communication devices.

Finally, as opposed to the other methods, we demonstrate the feasibility of the DDQN-based algorithm. There are one offloading action algorithms constructed using a well known greedy approach, as described in [20]. Remember that our recommended approach will be made based on the DDQN-based method, because of our rewards (Sum of delay, energy, and capacity). While depending on the data sizes transmitted (i.e., delays related), the greedy approach can be used to offload decision-making. We remind you that our proposed algorithm will be made based on the DDQN-based method because of our rewards.

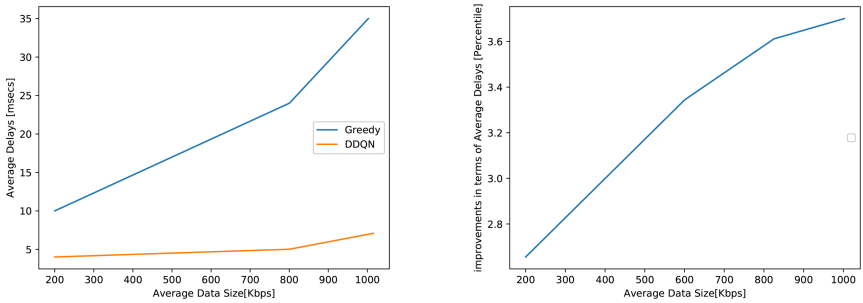

Fig. 4. Average delays and performance gap.

The delays increase the average data size for offloading increases. Our DDQN, the efficient policy will be taken during offloading decisions that consider delay, energy, and capacity, and the delays are lower than the greedy based decision making. Our proposed decision-making approach based on DDQN, as shown in Fig. 4, outperforms the greedy; and the gaps are increasing increasingly as the average data sizes increase.

6 Conclusion

We studied the issue of the allocation of resources and loaded balancing strategy in a dynamical system in mobile edge computing. While using high-quality, real-time communications in mobile edge networks. The problem is formulated under the minimal feasible delay and computational capability as the optimization problem. We used a DRL-based approach to find the optimum solution to deal with this issue. In the SDN controller, we have developed a smart agent to build an adaptive strategy for dynamic offloading and allocation of computing resources for multiple users as well as dynamic Video streaming information. The new DDQN-based agent requires the state system to determine the best strategy through interactions with the environment after the new MEC system framework. We have checked out a variety of experiments to show that the proposed research is realistic. The results of our simulation can be modified for performance evaluation due to environmental conditions or reward decision parameters. The results of the experiment show that the recommended system substantially decreases the cost of the system compared to the local, offloading, and greedy reference points.

References

1. Kim, J., Molisch, A.F.: Fast millimeter-wave beam training with receive beamforming. J. Commun. Netw. **16**(5), 512–522 (2014)
2. Kwon, D., Kim, S.W., Kim, J., Mohaisen, A.: Interference-aware adaptive beam alignment for hyper-dense IEEE 802.11 ax Internet-of-Things networks. Sensors **18**(10), 3364 (2018)

3. Kim, J., Caire, G., Molisch, A.F.: Quality-aware streaming and scheduling for device-to-device video delivery. IEEE/ACM Trans. Netw. **24**(4), 2319–2331 (2015)
4. Dao, N.N., Vu, D.N., Na, W., Kim, J., Cho, S.: SDCO: stabilized green crosshaul orchestration for dense IOT offloading services. IEEE J. Sel. Areas Commun. **36**(11), 2538–2548 (2018)
5. Tran, T.X., Pompili, D.: Joint task offloading and resource allocation for multi-server mobile-edge computing networks. IEEE Trans. Veh. Technol. **68**(1), 856–868 (2018)
6. Narendra, P.M., Fukunaga, K.: A branch and bound algorithm for feature subset selection. IEEE Trans. Comput. C **26**(9), 917–922 (1977)
7. Wang, T., Zhang, G., Liu, A., Bhuiyan, M.Z.A., Jin, Q.: A secure IoT service architecture with an efficient balance dynamics based on cloud and edge computing. IEEE Internet Things J. **6**(3), 4831–4843 (2018)
8. Bertsekas, D.P., Bertsekas, D.P., Bertsekas, D.P., Bertsekas, D.P.: Dynamic Programming and Optimal Control, vol. 1. Athena Scientific, Belmont (1995)
9. Park, S., Kim, J., Kwon, D., Shin, M., Kim, J.: Joint offloading and streaming in mobile edges: a deep reinforcement learning approach. In: 2019 IEEE VTS Asia Pacific Wireless Communications Symposium (APWCS), Singapore, pp. 1–4. IEEE (2015)
10. Mnih, V., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015)
11. Xu, Z., Wang, Y., Tang, J., Wang, J., Gursoy, M.C.: A deep reinforcement learning based framework for power-efficient resource allocation in cloud RANs. In: 2017 IEEE International Conference on Communications (ICC), Paris, pp. 1–6. IEEE (2017)
12. Tang, F., Fadlullah, Z.M., Mao, B., Kato, N.: An intelligent traffic load prediction based adaptive channel assignment algorithm in SDN-IoT: a deep learning approach. IEEE Internet Things J. **5**(6), 5141–5154 (2018)
13. Li, D., et al.: Deep reinforcement learning for cooperative edge caching in future mobile networks. In: 2019 IEEE Wireless Communications and Networking Conference (WCNC), Marrakesh, pp. 1–6. IEEE (2019)
14. Gazori, P., Rahbari, D., Nickray, M.: Saving time and cost on the scheduling of fog-based IoT applications using deep reinforcement learning approach. Future Gener. Comput. Syst. **110**, 1098–1115 (2019)
15. Hasselt, H.V.: Double q-learning. In: Advances in Neural Information Processing Systems 23, pp. 2613–2621. Curran Associates Inc. (2010)
16. Wang, C., Yu, F.R., Liang, C., Chen, Q., Tang, L.: Joint computation offloading and interference management in wireless cellular networks with mobile edge computing. IEEE Trans. Veh. Technol. **668**, 7432–7445 (2017)
17. Index, C.V.N.: Cisco visual networking index: Forecast and methodology 2015–2020. White paper, CISCO (2015)
18. Satyanarayanan, M., et al.: Edge analytics in the Internet of Things. IEEE Pervasive Comput. **142**, 24–31 (2015)
19. Cao, Y., Jiang, T., Wang, C.: Optimal radio resource allocation for mobile task offloading in cellular networks. IEEE Netw. **285**, 68–73 (2014)
20. Feng, W.J., Yang, C.H., Zhou, X.S.: Multi-user and multi-task offloading decision algorithms based on imbalanced edge cloud. IEEE Access **7**, 95970–95977 (2019)