INTELLIGENT TECHNIQUE FOR MULTI USER OFFLOADING IN MOBILE EDGE COMPUTING

Isha Bhargav[1] Raj Kumari[2]
University institute of engineering and technology, Panjab University
University institute of engineering and technology, Panjab University

ABSTRACT

Energy harvesting and mobile edge computing, both have the potential to increase the longevity of wireless data. In this paper, several mobile users are taken into account in a MEC setup which generates stochastic task arrivals and wireless channels, which are designed to jointly optimize latency and energy-efficient communications Computation is done, where the deep reinforcement learning-based dynamic algorithm decides whether to take offloading enabled, which is either computing in the MEC server, or partially computing to local devices, for each unit of time. For continuous action space, a deep deterministic policy gradient approach is used, which learn effective computation offloading policies individually for each mobile user. Hence, assign powers for both local execution and task offloading from the learned policy This study confirms that the suggested dynamic algorithm achieves the expected performance according to the performance evaluation using data-intensive simulations as compared to the traditional deep reinforcement learning algorithm.

Keywords: Deep reinforcement learning, extended Deep Q-learning for offloading decision, mobile edge computing

1. Introduction

As mobile user equipment (UEs) like smartphones, tablets, and Internet of Things (IoT) devices become significantly more popular, new mobile apps like navigation, facial recognition, and interactive online gaming are continually appearing. But these mobile devices have a limited amount of computation capacity and are energy-hungry applications, which restricts them from providing quality of service(QoS)[1].In order to increase service availability, speed, and reliability, mobile cloud computing(MCC) is introduced which facilitates compute and storage transfer from resource-constrained mobile devices to cloud servers. Additionally, it can lower the device's energy use without affecting how well applications run[2].MCC provides hardware as well as software services to run this computation-intensive application on the cloud server, this reduces cost and provides more scalability.

With recent advancements in IoT are resulting in more demands that are unable to meet by MCC, few requirements are geo-distribution, low latency, position awareness, and mobility support[3]. Traditional cloud computing solutions let UEs use the tremendous computational capacity in distant public clouds, but they may result in lengthy delays because of data interchange across wide area networks (WANs). While the wireless spectrum resource has been fully utilized in MCC through the use of techniques like Ultra Dense Networks (UDN) and Dynamic Spectrum Access (DSA), the big data traffic brought on by computationally intensive tasks grows exponentially and frequently goes unaddressed due to a lack of an appropriate resource assignment scheme to handle the massive service traffic[4]. Cloud computing may be used for computation-intensive services to compute activities on the cloud but at the expense of transmission and information leakage. By assisting nearby computational access points (CAPs) in networks with

computing chores, mobile edge computing (MEC) has been offered as a solution to this issue. MEC may drastically reduce latency and energy usage for both communication and calculation[5].

MEC is a novel paradigm in Cloud radio access networks that can increase the compute capacity at the edge of mobile networks by adding high-performance servers[6].MEC is the extended cloud that is positioned at the periphery of mobile networks, removing the need to transfer resource-intensive computing and storage operations from mobile devices to the network's core (the centralised cloud data centre)[7]. Offloading tasks eases the computational load on mobile devices and extends battery life. By using machine learning and deep learning techniques a system is trained to automatically make decisions on whether to compute tasks locally or server[8]. It is important to have effective resource management, The goal of a MEC system is to handle as many compute offloading requests from various MDs as is feasible given the system's constrained processing and radio capabilities. The system might result in sluggish reaction times and excessive power consumption if resource allocations are not coordinated well, which would then have an impact on the performance of the overall computation offloading.

This proposes an intelligent offloading decision for handling the optimization problem of computation offloading and resource allocation for a dynamic and continuous MEC environment.

Key contributions can be summarized below

- A multi-user system is considered, where each user learns a dynamic continuous offloading policy with the help of random task arrivals and time-varying wireless channels. The main objective is to optimize the cost sum with respect to power consumption and delay.
- An approach is adopted in which each user will independently develop a decentralized dynamic computation offloading policy, that determines an action, i.e., allocates powers for both local execution and computation offloading, based on its observations of the environment.
- Simulations are performed and compared with baseline strategy, and policy is learned according to DDPG. An offloading decision is made which determines whether to fully offload the data or partially offload the data to the MEC server or to a local device.

2. Related works

The advantages of computational offloading in Mobile Cloud Computing (MCC), such as energy savings and increased mobile application performance, have drawn attention[9]. But more advancements in technology have led to problems in cloud servers to provide efficient services for computation-intensive tasks, which is overcome using mobile edge computing. [4]have proposed energy-efficient computation offloading, which minimizes energy for all users using mixed integer non-linear programming problem (MINLP) and provides a value iteration based Reinforcement Learning (RL) technique, known as Q-Learning, to ascertain the ideal plan of joint computation offloading and resource allocation. In [5] a multiuser mobile edge computing (MEC) network enables users to offload some of their duties to several computational access points (CAPs). That takes into account real-world scenarios where task characteristics and processing capacity at the CAPs may change over time, posing an issue with dynamic offloading. Which first frame this issue as a Markov decision process (MDP) and then introduces the state and action spaces to address it. The users may dynamically fine-tune the offloading proportion in order to assure the system performance as evaluated by the latency and energy consumption. They also build a unique offloading approach based on the deep Q network (DQN). [10]have taken into account two user MEC networks, which have a series of tasks to execute, where the result of the task at WD1 is needed to compute an intermediate task at WD2. A mixed integer optimization problem is introduced to reduce the weighted total of the WDs' energy consumption and task execution time. This is done because of the inter-user task interdependence. Each WD's transmission power, local CPU frequencies, and workload offloading choices are cooperatively optimised. For multi-access edge computing (MEC) networks, [11] have novel deep imitation learning (DIL) driven edge-cloud computation offloading system has been developed. This approach uses optimal behavioral cloning to minimize the offloading cost of a time-varying

environment. Supervised learning from demonstrations to observation is carried out through behavioural cloning. For the intelligent framework, they have created a deep imitation learning-based offloading model, which is initially trained offline using learning examples. The model generates near-optimal online offloading choices with a very quick inference speed after a simple and quick implementation.[12] Analyze a combined resource allocation and computation offloading problem using multiple user equipment (UE) that is EH device and rechargeable battery equipped. The goal of the task is to reduce system energy usage while ultimately fulfilling the UE's latency limitation. there are three steps used for the formulation of an intractable mixed integer nonlinear programming (MINLP). Which initially uses a deep reinforcement learning framework called Deep Deterministic Policy Gradient to get continuous power allocation (DDPG). After that, channel assignment is determined using the Lagrangian function and the Karush-Kuhn-Tucher (KKT) condition. Finally, we change the DDPG framework's state, action, and reward. According to the simulation results, our suggested method discovers offloading decisions and power allocation with the lowest energy usage. [13] investigate partial computation offloading by jointly maximising the smart mobile device's (SMD) compute speed, transmit power, and offloading ratio with two system design goals: minimising the SMD's energy consumption (ECM) and minimising the delay of the application's execution (LM). Both the ECM issue and the LM problem are considered nonconvex problems, taking into account the scenario where the SMD is serviced by a single cloud server. A variable substitution approach is used to recast the ECM issue as a convex one and find the best solution. A locally optimum solution using the univariate search technique is suggested to handle the nonconvex and nonsmooth LM issue. Additionally, the scenario is expanded to include a system with many cloud servers, allowing the SMD to outsource its computation to a number of cloud servers. In this case, they are able to determine the ECM and LM issues' optimal cloud server distribution of computation in closed form. Finally, thorough simulations show that, compared to the current offloading systems, suggested algorithms dramatically cut energy usage and shorten the delay.

3. Preliminaries

3.1 Markov decision process

Markov decision process (MDP) basically formalizes the sequential decisions making process. MDP consist of five components that are agent, environment(E), state(S), action(A), and reward(R). The agent is the main decision maker who continuously interacts with the environment. It follows a stochastic deterministic policy (P: $P(a/s)$) where $a\epsilon A, s\epsilon S, r\epsilon R$, that maps the current state to action with the target to achieve maximum rewards, it is the probability distribution of action of a given state. The environment is a transition dynamic that how the state in the environment changes and how action gets reward is the joint distribution of the next state and reward based on current state and action applied by the agent[1].

$$P(s^i|s,a) = \sum_{h\epsilon R} P(s^i, r|s, a) \quad (a\epsilon A, s\epsilon S, r\epsilon R) \tag{1}$$

Policy function defines what action should be taken at by agent in particular state. The state transition probability ($P(s^i|s,a)$) which is defined as state transition and reward marginalization that is probability of reaching s if action a is taken in state s^i .State is defined as action should be taken to get maximum reward. The total discount return(γ), essentially determines how much the reinforcement learning agents cares about rewards in the distant future relative to those in the immediate future, and range between 0 and 1, and gives more importance to the initial rewards. The weighted sum of reward($G_{,t}$) is defined as

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 + R_{t+2} + \ldots + \gamma^{N-1} + R_{t+N} \tag{2}$$

The value of state ($V_{\Pi}(s)$) which is total expected discount with respect to a policy Π

$$V_{\Pi}(s) = E_{\Pi}[G_t|S_t = s] \tag{3}$$

Suppose at *S,* policy is applied, the value of the state is Equation(3) ,by substituting the value from Equation(2) in Equation(3) we get the recursive equation of policy, transition dynamics and successive state

$$V_{\Pi}(s) = \sum_a \Pi(a|s) \sum_{s^i,r} P(s^i, r|s, r)[r + \gamma\, V_{\Pi}(s)] \tag{4}$$

For a particular state the optimal state value is the maximum state value $V_*(s) = \max V_{\Pi}\,(s)$. For a particular state the optimal action is $q_*(s,a) = max_{\pi} q(s,a)$.To achieve maximum rewardThe optimal bellman equation is applied to the best optimal policy to get the maximum reward. For any policy Π according to Equation(3)

$$V_{\Pi}(s) = E_{\Pi}\,[G_t|s_t = s]$$

If policy Π is optimal

$$V_*(s) = max_a\, E[R_{t+1} + \gamma V_*(s^i)|S_{b=s^i} A_b\, = a] \tag{5}$$

Bellman optimality equation for state value is the summation of the sum of reward, the discount optimal value for the successive state[28]

$$V_*(s) = max_a \sum_{s^i,r} p(s^i, r|s, a)[r + \gamma\, V_*(s^i)] \tag{6}$$

For maximisation reward two algorithm is used which policy iteration algorithm and value iteration algorithm. Policy iteration algorithm is performed in two steps which are policy evaluation which compute values for state using policy, this policy is provided using policy improvement that improve the policy to get high state values. Figure1 shows policy iterative process ,firstly, a random policy is selected which is evaluated by policy evaluation that calculate state function for that particular policy. Using policy improvement, the state function is improved and next policy is achieved, this process continues for n number of times.

$$\pi_0 \rightarrow V_{\pi_0} \rightarrow \pi_1 \rightarrow V_{\pi_1} \rightarrow \pi_n \rightarrow V_{\pi_n}$$

Figure 1 Policy iterative process

Algorithm1:Policy iterative algorithm

Input: assumed states

Output: iterative policy

1.Set arbitrary value for the state

2.Using policy Π compute new value for the states

$$V_{new} \leftarrow \sum_{s,r} P(s^i, r|s, \pi(s)[r + V(s^i)] \tag{7}$$

3. repeat(2) until convergence of the state value.

4.Policy improvement

5.For all data sϵ S, update Π(s) as

$$\Pi(s) \leftarrow argmax \sum_{S^i,r}^{i} P(s^i, r|s, a)[r + \gamma V(s^i)] \tag{8}$$

In policy improvement it takes only action that maximises the summation of next reward and discounted factor with value of next state as shown in Equation(8)

3.2 Reinforcement learning

MDP derives optimal policy implicitly from the environment .RL does the opposite of MDP it optimise the predicted discounted benefits, the agent learns from its actual interactions with the environment and adjust its behaviour as a result of what happens. The bellman optimal equation is defined as

$$V^*(s_t, a_t) = \Pi[r(s_t, a_t) + \gamma max_{at+1} V^*(s_{t+1}, a_{t+1})] \quad (9)$$

This Equation(9) is updated using agent's experience tuple(s_t,a_t,r_t,s_{t+1}) and at time step *t* the other learned estimates are as follows

$$V(s_t, a_t) \leftarrow V(s_t, a_t) + \alpha\,[r(s_t, a_t) + \gamma max_{at+1} V(s_{t+1}, a_{t+1}) - V(s_t, a_t)] \quad (10)$$

Where α is the learning rate and Equation(10) can be called as Q-value for Q learning. It is off-policy which directly generates the optimal Q-value.

3.3 Deep Q learning

The traditional method for calculating finite MDP is Q learning algorithm, which is an off-policy RL algorithm**.** For large-scale problems, a deep Q network is used, which is the function approximate of Q learning, that sustains off-policy learning with deep neural networks.[1]**.** For self-learning from experience, DQN uses replay memory and uses a target network to minimize the relation between the recent model estimate and target value and observation. With the use of a target Q network, the network is trained to provide reliable targets during backups with temporal differences[20]. The experienced value is stored(s_t,a_t,r_t,s_{t+1}) in buffer B at time t. A target network whose parameters are updated to match those of the online model every t step is used to train the network with parameters by sampling mini-batches (s, a, r, s′) ~ U(B) from memory. To reduce loss, the model is trained.

$$L(\Theta) = V(s, a, r, s) \sim m[(y^{DQN}{}_i - Q(s_i, +a_i; \Theta))^2], y^{DQN}{}_i = r_i + \gamma\, max_a Q(s^i{}_i, a; \theta^-) \quad (11)$$

Where m is uniform distribution and y_i^{DQN} is the target value. To improve stability DQN uses a soft update that tracks the weight of learned network $\theta' \leftarrow \tau\theta + (1 - \tau)\theta'$ with $\tau \ll 1$. This approach can only handle discrete, low-dimensional action spaces it cannot solve issues in high-dimensional observation spaces. There are several interesting problems that feature continuous (real-valued) and high dimensions action spaces, most notably physical control tasks[29]. DQN cannot be understood simply. As it depends on identifying the course of action that optimizes the action-value function, which in the continuous-valued scenario necessitates an iterative optimisation approach at each stage. Hence there require a technique which can manage continuous high-dimensional action spaces. Deep deterministic policy gradient (DDPG) is the technique which aims to maximize rewards for continuous offloading decision

3.4 Deep deterministic policy gradient

Since finding the greedy policy in continuous action spaces necessitates an optimization of at each timestep, it is not possible to directly apply Q-learning to these problems. This is because large, unconstrained function approximators and nontrivial action spaces necessitate an optimisation that is too slow to be practical [30] . In place of it, an actor-critic strategy based on the DDPG algorithm can be used. The deterministic term is stochastic which policy network under some observation gives the most accurate action. It is a model-free actor-critic approach, which learns policy in low dimension and in continuous

action space. In an actor-network, it takes observation and gives action. In a critic network, it takes observation and gives a Q value which measures how good the action is

$$critic = Q_{reward} + \gamma . Q_{next} \tag{12}$$

DDPG is off policy and it consists of replay before so that it can be trained on real-world experience[25]. Figure 2 describes the architecture of DDPG, The replay buffer recalls a series of observations, actions, rewards and the next observation. For continuous action space, the policy of action is set by treating Equation(10) as an optimization problem for every action and solved with gradient descent. Training of neural network to get an optimized problem is done using two DNNs, where the state is given as input and action is received in input. Critic Q(s, a) works similar to DNNs and update Equation(11), learning is done according to the bellman equation in Q learning. Actor $\mu(s, a)$ deterministically links a state's state to a certain continuous action, while updating the expected return from the start distribution J with regard to the actor parameters by using the chain rule

$$\Delta\mu J \approx V(s, a, r, s^i) \sim U(B)[\nabla_a(s, a|\theta^q)\nabla\theta^u \mu(s|\theta^\mu)] \tag{13}$$

Here critic is updated using following equation $\theta^Q \leftarrow \theta^Q$-$\alpha_Q \nabla\theta^q$.L($\theta^Q$) and actor is updated using $\theta\mu \leftarrow \theta\mu$\-$\alpha_\mu \nabla\theta^u$J where α_Q and α_μ are learning rates.

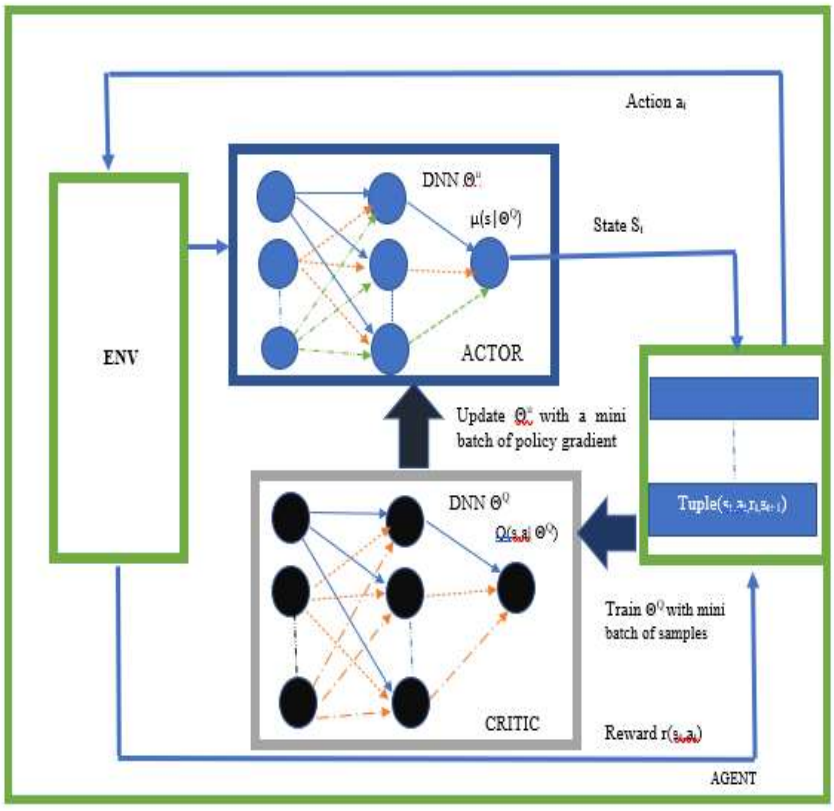

Figure 2 Architecture of DDPG

3.5 System model

In this section, the MEC system is as shown Figure 5, with one small cell which consists of multiple mobile users and one base station. All users are assumed to have computation-intensive tasks to complete. MEC servers are heterogeneous, these are deployed close to the base station along with mobile users. A discrete-time model is used, in which the operational period is slotted with an equal length of time slots and indexed by T=.{0,1,2,3…N} Each user's task arrival and channel state differ at each slot t ∈T. Therefore, each user must decide the ratio of local execution and compute offloading at each slot in order to balance the average energy consumption and task processing latency[31], [32].

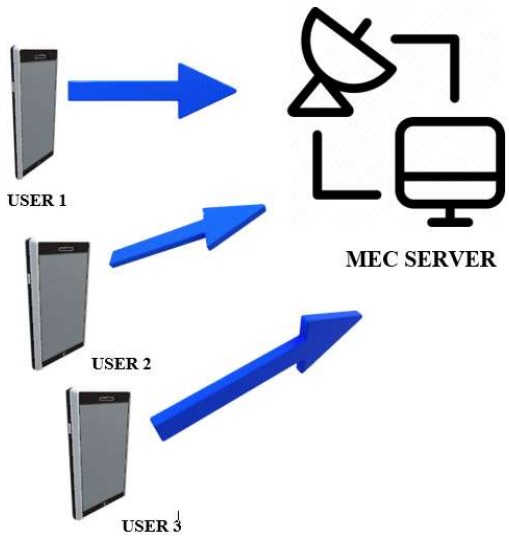

Figure 3 System model for offloading decision of multiple users

3.6 Task generation model

MEC systems, which consist of N antennas, at each time slot t ∈T, the channel vector of each user n ∈N is represented by $h_n(t)$. This channel vector has several attributes called tasks. These tasks are defined as *{id, model,num_r, rate,*d_f*,* t_s*,* S_d*,* A_c*,p},* where *id* defines the task generated by the particular user. model is the AR *model,num_r* is the number of rewards, d_f is the data buffer size,t_s is the tradeoff factor between energy consumption and buffering delay. S_d is the state dimension, A_c is the action dimension and p is the penalty. Each user has different task arrivals during time slot t∈T. For each time slot t∈T, the received signal is defined as

$$R_x = \sum_{n=1}^{n} h_n(t)(p_o)^{0.5}V(t) + n(t) \quad (14)$$

Where p_o it is the maximum value of power transmitted to offload bits by user n∈N. V(t) is state space and n(t) is the white Gaussian noise. The signal-to-noise ratio is defined by

$$No(t) = \frac{p_o(t)}{\sigma^2 r[h_n(t)]} \quad (15)$$

The uplink rate for one user is defined as R [33]

$$R = \frac{W}{K} log(1 + \frac{p_o G}{\frac{W}{K} No}) \quad (16)$$

3.6.1 Computation Model

Each user takes advantage of local execution or computation offloading in this section. To compute the task arrivals at slot $t \in T$, $k_n(t)$ is the number of task arrival of user n, which is distributed over different time slots. It is considered that bits denoted by $B_{o,n}(t)$ are either computed locally or transferred on the MEC server. $Q_{,n}(t)$ is the queue length of user n's task buffer at the initial slot t. This is defined as

$$Q_n(t+1) = [Q_n(t) - B_{o,n}(t)] + K_n(t) \quad \forall\, t \in T \tag{17}$$

3.6.2 Local computing

Heterogeneous computing capability is considered, bits can be processed locally if the device has sufficient computing capacity[34]. By adopting the dynamic voltage and frequency scaling approach, which dynamically aims to reduce the power consumption and adjust voltage and frequency of CPU. ৶(t) is the maximum allowable frequency, which is the ratio of power consumption for local processing to *k* which is effective switching capacitance in the chip for slot t which is defined as

$$৶(t) = (p_{l,n}/k)^{0.33} \tag{18}$$

The local execution delay for a particular task T_n where ৶ is the CPU cycle overall computation frequency to run task $h_n(t)$

$$T_n^l = \frac{৶}{৶(t)} \tag{19}$$

Corresponding energy consumption of task $h_n(t)$ which is defined as

$$E_n^l = k(৶(t))^2 T_n \tag{20}$$

$k(৶(t))^2$ is the energy consumption per CPU cycle to complete task $h_n(t)$. According to practical observation [35]

The trade off between energy consumption and delay is described as total energy consumption and local execution delay.

$$cost_l = w_n T_n^l + (1 - w_n^t) E_n^l \tag{21}$$

Where $w_m{}^t$ and $w_m{}^e$ weights of energy and time cost of task $h_n(t)$. The weights are assumed and process as $w_m t + w_m{}^e = 1$.

3.6.3 Offloading computing model

If the task initiated by the user is large, the data cannot be executed on mobile devices as there are restricted capabilities of local computing. This generated task is offloaded to the MEC server. The tradeoff between energy and delay is calculated. The transmission delay(T_n^f) is the time required to offload data to the MEC server is defined as.

$$T_n^f = \frac{D_b}{R} \tag{22}$$

Where D_b is the size of the input data and R is the uplink rate. The corresponding energy($E^f{}_m$) is defined as

$$E_n^f = P_o T_n^f + D_b E_e \tag{23}$$

E_e energy consumed by each bit is calculated by the servers on the base station. The total cost of energy and delay is described as [5]

$$cost_o^f(C) = -[w_n^t T_n^f + (1 - w_n^t) E_n^f] \tag{24}$$

In order to guide agent to learn the experience of minimizing response time and energy consumption, negative of weighted sum of transmission delay and energy consumption is used.

3.7 Offloading decision

Since both MEC server and local device have limited computation capacity, and user requirements are high. There require a way with which users have a high transmission with a limited amount of energy and less delay[36]. We have proposed a method through which according to user requirements computation, offloading decision is done, which is discussed in algorithm 2.C_1 and C_2 are threshold values which are derived from Equation(24).C1 is the maximum optimized cost ,C2 is minimum cost of user on their local device. For each user cost C_i is considered which compared with these parameters and offloading decision is made.

Algorithm 2: Computation offloading decision
Input: offloading parameter
Output: offloading decision
1 if the task is decided to offload **then**
2 if $C_i > C_l$ **then** // **C_i is the cost of particular user and C_l maximum optimized cost**
Offload the task to edge server
3 end if
4 else if task is decided to offload partially or locally **then**
5 **if** $C_2 < C_i < C_l$ **then** //**C_2 is the minimum optimized cost**
6 tasks is partially offloaded i.e some bites to the mobile device and some on MEC server
7 **else**
8 tasks is computed on the local device.
9 **end if**
10 end if

3.8 Computation offloading decision using deep reinforcement learning

For MEC system's resource allocation and offloading, a DRL-based technique is used to reduce the computing costs of each mobile user in terms of power consumption and delay. A deep deterministic policy gradient(DDPG) is adopted, and each user will independently develop a dynamic computation offloading policy, which determines an action, i.e., allocates powers for both local execution and computation offloading, based on its observations of the environment. Since no user has any prior knowledge of the MEC system, no user agent is aware of the M number of users, task arrival data, or wireless channel numbers. As a result, the online learning process is completely model-free. This framework is introduced, where state space, action space, and reward are described [20].

3.8.1 State Space

The system consists of two components: channel vectors and queue lengths of the buffer for all users. This information is gathered by BS and then dispersed to every user [37]. The state of each user is determined by total observation of the system, in which the selection of action is done independently.
At time *t,* the queue length of each user m task buffer is adjusted, and simultaneously an acknowledge signal is sent from the BS which is the last SINR of user n. At the same time, channel reciprocity may be used to predict the channel vector $h_m(t)$ for the impending uplink broadcast. Hence the state space can be defined as

$$V(t) = \{Q(t), P_r(t), h_m(t)\} \tag{25}$$

At time *t*, power ratio P_r after noise detection at BS is defined as

$$P_r(t) = \frac{No(t)\sigma^2 r}{p_o(t)||hm(t)||} \tag{26}$$

3.8.2 Action Space

According to V(t), the current state is observed by each user n. An action $a_m(t)$, allocates all the power of local computing and MEC offloading at time slot *t*.

$$a_m(t) = \{p_l, p_o\} \tag{27}$$

Where $p_l(t) \in \{0, p_l(t)\}$ and $p_o(t) \in \{0, p_o(t)\}$.It is worth noting that, unlike other traditional DRL algorithms, this approach chooses from a set of predetermined discrete power levels in order to reduce the average computing cost. Consequently, it is possible to lower the high dimension of discrete action spaces drastically.

3.8.3 Reward

This model considers the trade-off between energy consumption and delay, which is by minimization of energy while fulfilling the task at an adequate delay.
Hence the total sum of cost is the summation of total energy cost with penalty delay.

$$R = Wm(Cost^{l} + Costo^{f}) \quad (28)$$

3.9 Training and testing

There are three stages for an offloading technique to learn and assess, which are the data generation phase, training phase and testing phase. In the first stage, each mobile user interacts with the simulated environment, which imitates user behavior with the MEC systems and returns an acknowledgement bit consisting of CSI and SINR
In the second stage, training is done, and the training steps are discussed in Algorithm 3. Communication between the user and the MEC environment is a contiguous RL task, which is initiated by manually starting the state $V_{n,1}$ with maximum steps T_{max} , for each episode. During each episode at time step *t,* each episode will store a tuple value$\{V_t,s_{1,t},R,V_{t+1}\}$ in buffer V_t. On other hand, a mini-batch consisting of experienced tuple value$\{V_t,s_{1,t}, R, V_{t+1}\}_{t=1}^{l}$ is used to update the actor and critic network.
At the testing stage, each user will load its actor networks from the training phase. After which, the user will empty the data buffer and connect with the randomly generated environment. When its local observation of the environment is gained as the current state, it then chooses actions based on the output of the actor-network.

Algorithm 3: Training of DDPG
Input: randomly generated actor, critic and target network
Output: maximum number of rewards

1. for each user agent, nϵN **do**
2. Randomly generate network which is actor and critic
3. Set weight for target network
4. Set the replay buffer
5. for each episode 1,N **do**
6. Reset the MEC environment
7. Randomly generate the initial state for each users
8. for t=1,N **do**
9. Calculate power $P_{o,}$ select random action a_t and generate exploration noise
10. Perform action a_t emulator and observe rewards along with next state V(t+1) in the emulator
11. Collect and save $(V(t),a_t,R,,V(t+1))$ in the buffer L_n
12. Randomly sample mini batch of T tuples from L_n
13. Perform gradient descent step and update critic actor network.
14. Update target vector
15. End
16. End

4. Numerical results

In this section, numerical simulation is discussed. For simulation, the parameters considered in the proposed algorithm are described in the Table 1.

Table 1 Parameters for proposed algorithm

PARAMETERS	DESCRIPTION	VALUE
d_m	Distance between user n to base station	200 m
p_m	Channel correlation coefficient	0.95
W	Bandwidth	1MHz
F	Computation capacity of MEC server	5GHz
f_n^l	CPU frequency	1.26GHz
$P_{l,m}$	Max power required for local execution	2W
w_o	Decision weight	0.5,0.9
N	Number of users	3,5
N_o	Noise power	10^{-9} W
MB	Mini batch size	64
ɣ	Gamma	0.99
∝	Learning rate	0.0001
τ	Target Networks	0.001
k	CPU cycles per bit	10^{-27}
	Buffer size	25000
	Optimizer	Adam
	Activation function	Relu

For the execution of DDPG, four-layer fully connected neural network with two hidden layers of neurons 400 and 300 are considered[20]. Every layer consists of an activation function Relu and the last layer consists of the activation function sigmoid, which converts actions into ranges of 0 and 1. For the learning neural network parameters, the adaptive moment estimation(Adam) method is used with a learning rate of 0.001.There are 5 runs for numerical simulation with index of 10000 and episode length 1000.
The proposed algorithm is compared with the baseline algorithm and the offloading decision is made. Every user's behaviour is observed with respect to energy and rewarded with respect to noise. The goal is to find optimal policy that maximizes the long term expected discount reward it receives. For each time slot t, the channel condition and task arrivals varies. The objective is to balance the average energy and task processing delay, ratio of local execution and computation offloading at each slot for each users is computed.
In this section the analysis is done with respect to number of users. Three and five number of users are considered on which DQN and DDPG algorithm is applied. For three number of users the average reward received per episode is increases as the interaction between MEC and mobile device continues. As described in Table 2 and figure 4 ,when the episode index increases the average rewards per episode becomes stable, which indicates that policy learned for different user is learned correctly

Table 2 Average reward per episode for 3 number of users

AVERAGE REWARD PER EPISODE			
Algorithm	User1	User2	User3
DDPG	-18.15	-36.31	-81.48
DQN	-6.806	-11.21	-43.37

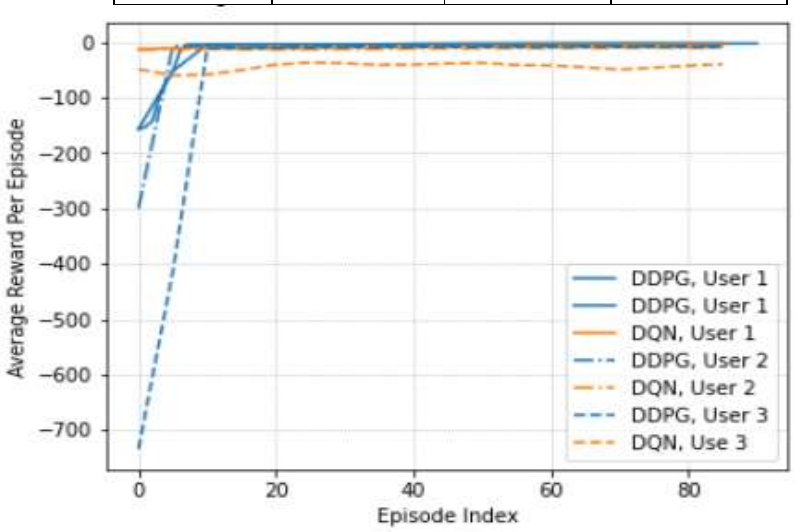

Figure 4 Average reward per episode for 3 number of users

The policy learned and average reward received per episode have direct relation with power consumption. DQN algorithm shows more efficient results as compared to DDPG algorithm. As the episode index is increasing the average power per episode becomes stable, as shown in Table 3 and Figure 5 .

Table 1 Average power per episode for 3 number of users

Algorithm	User1	User2	User3
DDPG	0.45	1.28	2.11
DQN	0.66	1.12	1.76

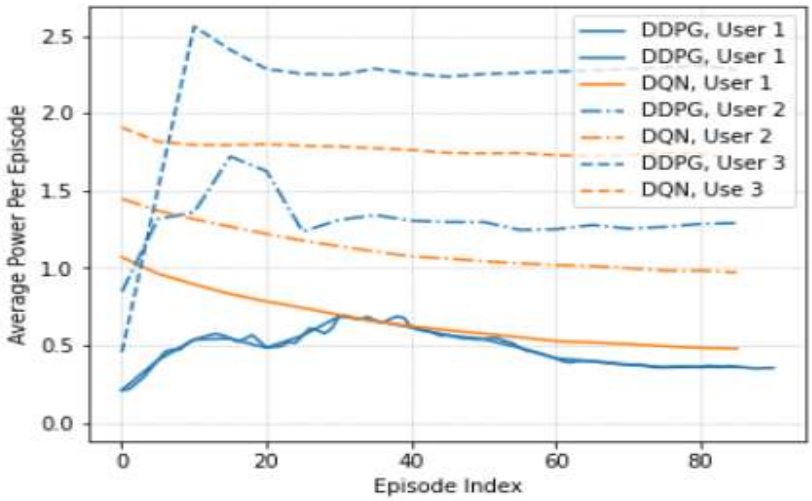

Figure 5 Average power per episode for 3 users

By setting the trade off factor w=0.9 energy is taken as priority by sacrificing delay, as there is limited battery life of mobile device. As the episode index increases the average delay per episode is more. For user 1 and user 2 DQN algorithm have less delay as compared to DDPG algorithm. As shown in Table 4 and Figure 6 we can say that DQN algorithm is more superior as compared to DDPG algorithm

Table 4 Average delay per episode for 3 number of users

Algorithm	User1	User2	User3
DDPG	21.08	40.34	96.14
DQN	2.59	20.73	145.59

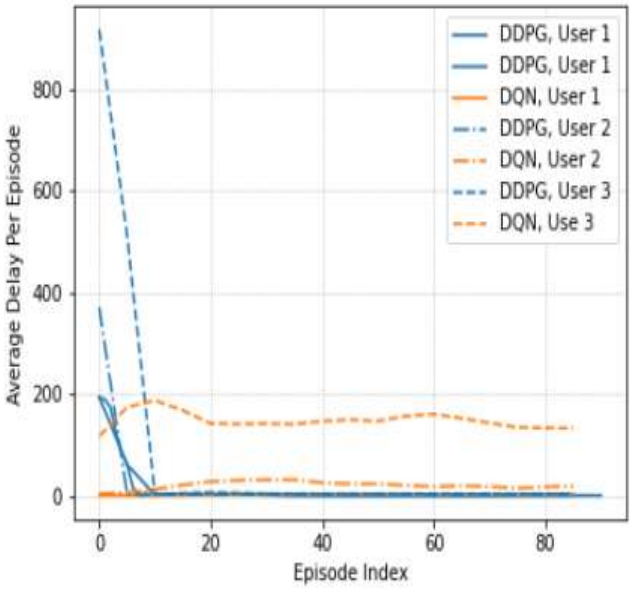

Figure 6 Average delay per episode for 3 number of users

For five number of users ,the penalty received for DQN algorithm and DDPG algorithm is increases as episode index increases, which indicates that computation offloading policies for different users with

different scenarios can be learned more efficiently ,with interaction between MEC server and mobile user. For DDPG algorithm the number of penalty per episodes increases, but becomes stable after episode index 20. As shown in Table 5 and Figure 7 more number of users there is high optimised cost and which results in high computation demand

Table 5 Average reward per episode for 5 number of users

ALGORITHM	User1	User 2	User3	User4	User5
DDPG	-15.94	-47.79	-125.17	-125.17	-125.17
DQN	-8.46	-13.08	-38.16	-38.16	-38.16

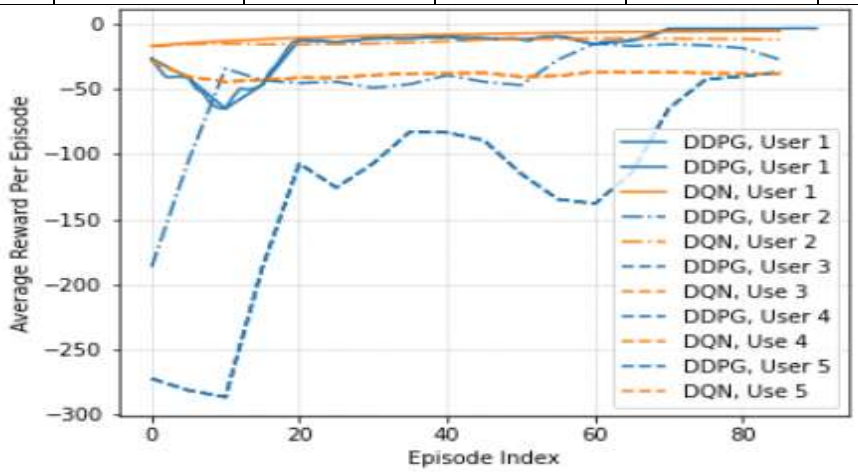

Figure 7 Average reward per episode for 5 number of users

It is observed that rewards received per episode have a direct relation with the average power per episode of each user. Since the penalty received are constant For DQN algorithm, the power consumed by each user is almost constant. In case of DDPG the number of rewards received were becoming stable after episode index 20,hence power consumption become stable after episode index 20. With more task arrival rate it is observed that DDPG perform less efficiently as compared to DQN as more power is consumed as number of users increases as shown in Table 6 and Figure8 .

Table 6 Average power per episode for 5 number of users

ALGORITHM	User1	User 2	User3	User4	User5
DDPG	0.246	0.756	1.262	1.262	1.262
DQN	0.637	1.068	1.499	1.499	1.499

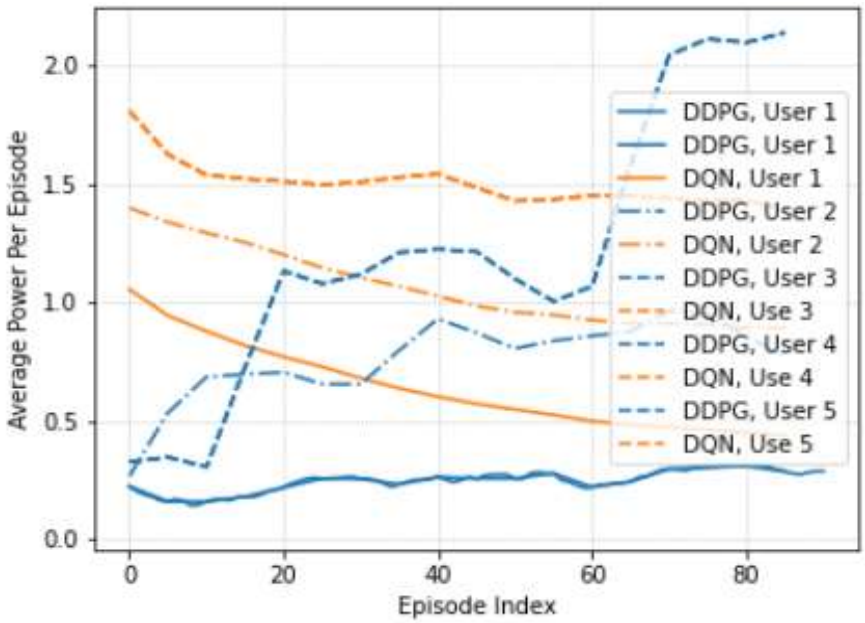

Figure 9 Average power per episode for 5 number of users

There is inverse relation between power and delay. Average delay per episode is minimum and less than 150 for each episode index as shown in fig. There is decreases in average delay per episode. For DDPG with most the penalty received per episode there is more delay as compared to DQN algorithm as shown in Table 7 and figure 10.

Table 7 Average delay per episode for 5 number of users

ALGORITHM	User1	User 2	User3	User4	User5
DDPG	84.08	238.19	587.89	587.89	587.89
DQN	3.44	29.39	246.32	246.32	246.32

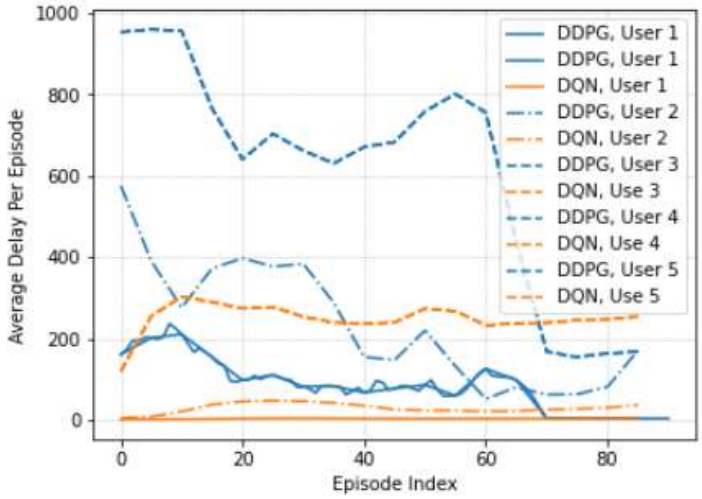

Figure10 Average delay per episode for 5 number of users

From the above observation we can say that DQN algorithm is a value based learning algorithm and tries to predict Q values for each state action pair in a single node, which learns the Q values from the defined policy. DDPG has a critic model that determines the Q value but uses the actor model to determines the action to take, hence DDPG algorithm tries to directly learn the policy. In continuous action space there is no meaningful way to produce finite set of actions. DQN algorithm are difficult to process continuous action space for more number of users, as the policy learning is very time consuming. Its output is deterministic and cannot solve random strategy efficiently. The degradation of the strategy can easily cause the algorithm to not give optimal solution. For less number of user the policy learned by both the algorithm is not optimal as compared to more number of users.

As the number of users n Figure 11 ,when the number of users is increasing there is a gradual increase in optimized cost when the number of users increases. The proposed DDPG gives more efficient results as compared to DQN with a minute difference in their performance. The optimized cost is almost constant when users are less than five, but there is a gradual increase in cost after the number of users increases. The reason for this is there is the limited capacity of the MEC server.

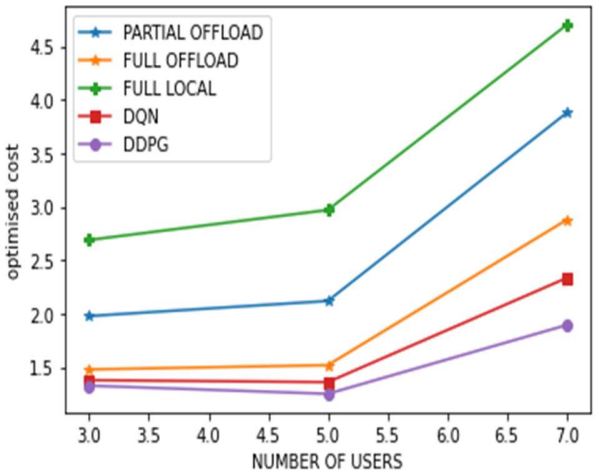

Figure 11 Number of users with respect to optimised cost

In Figure 12 , optimized cost with respect to data bites is considered, which increases as data bites increase. As more energy is consumed with more delay for offloading, DDPG has more prominent results as compared to DQN. DQN and fully offload have almost similar behaviour when task bites size increases. Fully local curve increases more rapidly as compared to other methods, which depicts that partially offloading the data or fully offloading will be the appropriate solution to save energy consumption.

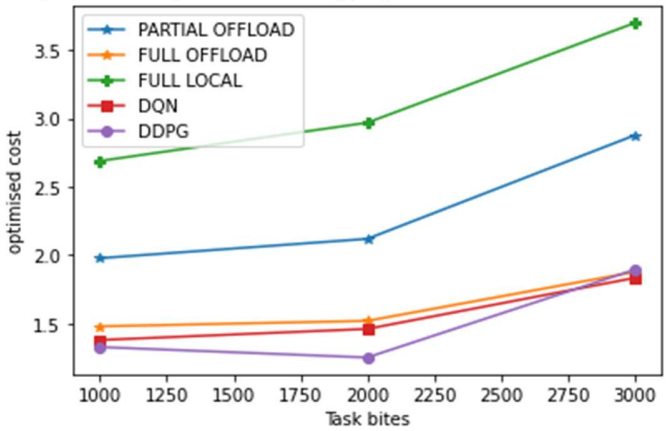

Figure 12 Optimised cost with respect to task bites

Acknowledgment

The author would like to express sincere gratitude to Dr. Raj Kumari for there support and valuable suggestions that helped and improve this paper. I would also like to pay gratitude to all the researchers who have given there contributions on this topic

Conclusion

In this research paper, a data distributed approach has been implemented because MEC servers and local devices have computation capacity, which varies for every user. Using DDPG a policy is created which is trained with deep reinforcement learning to make offloading decisions, which is to either fully offload the data to a local device or MEC server or partially. This determination is done according to optimized cost, which is the sum of the minimized cost of energy and delay. The future scope of this approach is to create an advanced policy which can handle a huge amount of data in a lesser amount of time. An optimized algorithm can be proposed that can reduce the time for the iteration.

References

[1] B. Zhang, G. Zhang, W. Sun, and K. Yang, "Task Offloading with Power Control for Mobile Edge Computing Using Reinforcement Learning-Based Markov Decision Process," *Mob. Inf. Syst.*, vol. 2020, 2020, doi: 10.1155/2020/7630275.

[2] P. Nawrocki and B. Sniezynski, "Adaptive Service Management in Mobile Cloud Computing by Means of Supervised and Reinforcement Learning," *J. Netw. Syst. Manag.*, vol. 26, no. 1, pp. 1–22, 2018, doi: 10.1007/s10922-017-9405-4.

[3] W. Zhang, Y. Wen, K. Guan, D. Kilper, H. Luo, and D. O. Wu, "Energy-optimal mobile cloud computing under stochastic wireless channel," *IEEE Trans. Wirel. Commun.*, vol. 12, no. 9, pp. 4569–4581, 2013, doi: 10.1109/TWC.2013.072513.121842.

[4] K. Jiang, H. Zhou, D. Li, X. Liu, and S. Xu, "A Q-learning based Method for Energy-Efficient Computation Offloading in Mobile Edge Computing," *Proc. - Int. Conf. Comput. Commun. Networks, ICCCN*, vol. 2020-August, 2020, doi: 10.1109/ICCCN49398.2020.9209738.

[5] C. Li *et al.*, "Dynamic Offloading for Multiuser Muti-CAP MEC Networks: A Deep Reinforcement Learning Approach," *IEEE Trans. Veh. Technol.*, vol. 70, no. 3, pp. 2922–2927, 2021, doi: 10.1109/TVT.2021.3058995.

[6] S. N. Shirazi, A. Gouglidis, A. Farshad, and D. Hutchison, "The extended cloud: Review and analysis of mobile edge computing and fog from a security and resilience perspective," *IEEE J. Sel. Areas Commun.*, vol. 35, no. 11, pp. 2586–2595, 2017, doi: 10.1109/JSAC.2017.2760478.

[7] Y. Jararweh, A. Doulat, O. Alqudah, E. Ahmed, M. Al-Ayyoub, and E. Benkhelifa, "The future of mobile cloud computing: Integrating cloudlets and Mobile Edge Computing," *2016 23rd Int. Conf. Telecommun. ICT 2016*, pp. 5–9, 2016, doi: 10.1109/ICT.2016.7500486.

[8] H. Eom, P. S. Juste, R. Figueiredo, O. Tickoo, R. Illikkal, and R. Iyer, "Machine learning-based runtime scheduler for mobile offloading framework," *Proc. - 2013 IEEE/ACM 6th Int. Conf. Util. Cloud Comput. UCC 2013*, pp. 17–25, 2013, doi: 10.1109/UCC.2013.21.

[9] W. Junior, E. Oliveira, A. Santos, and K. Dias, "A context-sensitive offloading system using machine-learning classification algorithms for mobile cloud environment," *Futur. Gener. Comput. Syst.*, vol. 90, pp. 503–520, 2019, doi: 10.1016/j.future.2018.08.026.

[10] J. Yan, S. Bi, Y. J. Zhang, and M. Tao, "Optimal Task Offloading and Resource Allocation in Mobile-Edge Computing with Inter-User Task Dependency," *IEEE Trans. Wirel. Commun.*, vol. 19, no. 1, pp. 235–250, 2020, doi: 10.1109/TWC.2019.2943563.

[11] S. Yu, X. Chen, L. Yang, D. Wu, M. Bennis, and J. Zhang, "Intelligent Edge: Leveraging Deep Imitation Learning for Mobile Edge Computation Offloading," *IEEE Wirel. Commun.*, vol. 27, no. 1, pp. 92–99, 2020, doi: 10.1109/MWC.001.1900232.

[12] J. Ren and S. Xu, "DDPG Based Computation Offloading and Resource Allocation for MEC Systems with Energy Harvesting," *IEEE Veh. Technol. Conf.*, vol. 2021-April, pp. 0–4, 2021, doi: 10.1109/VTC2021-Spring51267.2021.9448922.

[13] Y. Wang, M. Sheng, X. Wang, L. Wang, and J. Li, "Mobile-Edge Computing: Partial Computation Offloading Using Dynamic Voltage Scaling," *IEEE Trans. Commun.*, vol. 64, no. 10, pp. 4268–4282, 2016, doi: 10.1109/TCOMM.2016.2599530.

[14] A. Cini, C. D'Eramo, J. Peters, and C. Alippi, "Deep Reinforcement Learning with Weighted Q-Learning," 2020, [Online]. Available: http://arxiv.org/abs/2003.09280

[15] S. Park, D. Kwon, J. Kim, Y. K. Lee, and S. Cho, "Adaptive real-time offloading decision-making for mobile edges: Deep reinforcement learning framework and simulation results," *Appl. Sci.*, vol. 10, no. 5, 2020, doi: 10.3390/app10051663.

[16] J. Chen, H. Xing, Z. Xiao, L. Xu, and T. Tao, "A DRL Agent for Jointly Optimizing Computation Offloading and Resource Allocation in MEC," *IEEE Internet Things J.*, vol. 8, no. 24, pp. 17508–17524, 2021, doi: 10.1109/JIOT.2021.3081694.

[17] Z. H. Abbas *et al.*, "Computational offloading in mobile edge with comprehensive and energy efficient cost function: A deep learning approach," *Sensors*, vol. 21, no. 10, pp. 1–18, 2021, doi: 10.3390/s21103523.

[18] Z. Tong, X. Deng, F. Ye, S. Basodi, X. Xiao, and Y. Pan, "Adaptive computation offloading and resource allocation strategy in a mobile edge computing environment," *Inf. Sci. (Ny).*, vol. 537, pp. 116–131, 2020, doi: 10.1016/j.ins.2020.05.057.

[19] J. Li, H. Gao, T. Lv, and Y. Lu, "Deep reinforcement learning based computation offloading and resource allocation for MEC," *IEEE Wirel. Commun. Netw. Conf. WCNC*, vol. 2018-April, pp. 1–6, 2018, doi: 10.1109/WCNC.2018.8377343.

[20] Z. Chen and X. Wang, "Decentralized computation offloading for multi-user mobile edge computing: a deep reinforcement learning approach," *Eurasip J. Wirel. Commun. Netw.*, vol. 2020, no. 1, pp. 1–27, 2020, doi: 10.1186/s13638-020-01801-6.