Federated Deep Reinforcement Learning-based task offloading system in edge computing environment

Hiba Merakchi[1]
Ecole Nationale Supérieure d'Informatique (ESI ex.INI)
Algiers, Algeria
ih_merakchi@esi.dz

Miloud Bagaa[2]
Department of Electrical and Computer Engineering
Université du Québec à Trois-Rivières
Trois-Rivières, QC, Canada.
miloud.bagaa@uqtr.ca

Ahmed Ouameur Messaoud[3]
Department of Electrical and Computer Engineering
Université du Québec à Trois-Rivières
Trois-Rivières, QC, Canada.
messaoud.Ahmed.Ouameur@uqtr.ca

Adlen Ksentini[4]
EURECOM
Sophia-Antipolis, France.
Email: adlen.ksentini@eurecom.fr

Abdenour Sehad[5]
Ecole Nationale Supérieure d'Informatique (ESI ex.INI)
Algiers, Algeria
a_sehad@esi.dz

***Abstract*—Nowadays, Internet of Things (IoT) devices are gaining momentum globally. However, due to their limited size, these devices have limited battery capacity, computational resources, and wireless bandwidth, making it impossible to run resource-intensive applications on these devices. Fortunately, Edge Computing has emerged as a promising solution to meet this demand by enabling data processing in more capable devices. Task offloading is a crucial technique used in Edge Computing to overcome the limitations of IoT devices by offloading some of their computational tasks to more powerful edge servers. The traditional methods used for task offloading are often based on heuristics or simple rules, which may result in sub-optimal solutions. Moreover, the increasing complexity and heterogeneity of edge networks, as well as the stochastic nature of the wireless channel, pose significant challenges for these methods. In this paper, we leverage Federated Learning (FL) to efficiently train Deep Reinforcement Learning (DRL) agents to make the best offloading and power allocation decisions by achieving the near-optimal trade-off between task execution latency and the power consumption of the end device. The obtained simulation results of the proposed method demonstrate its remarkable and superior performance in comparison to central DQN.**

I. Introduction

Task offloading is a critical aspect of the efficient management of computing resources in modern networks, particularly in the context of the Internet of Things (IoT) and Edge Computing. As the number of connected devices and the volume of data generated by these devices continue to grow, the demand for computational resources proportionally increases. Task offloading provides a mechanism for transferring computational tasks from resource-constrained devices to more capable ones, such as Edge nodes, where the tasks can be executed promptly and efficiently. This allows for the efficient use of network resources, reduces energy consumption, and enhances the overall performance of the devices. However, traditional methods like heuristics and control theory-based algorithms have several limitations. One of the primary challenges is their limitation in the face of changing environments and unpredictable events, such as unexpected arrivals of new tasks, and the stochastic nature of wireless networks, which makes it difficult to predict and adapt to network conditions accurately. Additionally, these methods do not always lead to the optimal solution. The performance of traditional task offloading methods heavily relies on the selection of appropriate algorithmic parameters, which might require extensive manual tuning, which can be a time-consuming and impractical process. Furthermore, traditional methods may not be scalable to handle large-scale systems due to the computational complexity involved.

The promising results obtained by machine learning (ML) in various fields sparked our motivation to explore its potential to automatically, intuitively, and adaptively offload tasks in IoT networks. DRL can address some of the previously explained challenges and limitations by leveraging the power of neural networks to approximate complex functions and learn from previous experiences. By using an agent that interacts with the environment and receives feedback in the form of rewards, DRL algorithms can learn to make near-optimal decisions that optimize a specific objective, such as minimizing energy consumption and latency. Furthermore, these algorithms can adapt to changing environments and can handle the stochastic nature of wireless networks, resulting in more robust and efficient solutions.

In this work, we use federated learning (FL) to simultaneously train many agents in IoT devices to negotiate the near-optimal policy for task offloading and transmission power allocation in the Edge Computing environment. The proposed method will then be tested and evaluated in terms of achieved performance and convergence.

The remainder of the paper is organized as follows. We review the related work in Section II. The task offloading problem is formulated in Section III. Meanwhile, section IV describes the proposed solution. Section V presents the simulation results and their analysis, and finally, Section VI concludes the paper.

II. Related work

There has been considerable research on task offloading in wireless networks, particularly in the context of resource-constrained devices. The conventional task-offloading methods are usually based on game theory [1], linear regression [2], and dynamic programming [3]. These methods can only give an approximation to the optimal solution and are limited against dynamic task-offloading decision scenarios as they require prior knowledge of the environment. Most of the previously published studies optimized either the energy consumption or the task execution latency in the network [4], [5]. Other works have considered both [6], [7], while others aim to optimize other objectives like load balancing [8] and deployment cost [6], [9].

DRL has been shown to be effective in solving complex decision-making problems, such as task offloading and resource allocation in wireless networks. For example, the authors in [10] and [11] formulated the problem of task offloading for mobile users as a single-agent infinite-horizon MDP. The designed method aims to minimize the monetary cost and energy consumption of the end device. In [12], authors have used a single-agent ϵ-greedy Q-learning-based algorithm to achieve a better trade-off between the execution latency and the power consumption of both the end device and the edge server. However, their solution assumes that the channel conditions between the end devices and the gateway, the computation task queue, and the remaining computation resource of the device are the only network states that need to be considered for solving the task offloading problem.

There have been some efforts to combine FL and DRL in wireless networks. For example, in [13], Moon S et al. have proposed a Federated Deep Deterministic Policy Gradient (DDPG)-based offloading method with power control in vehicular edge computing (VEC). The authors used DDPG to handle the continuous actions that represent the level of the allocated transmission power for offloading to the VECS. In the work of [14], the authors propose a federated learning approach to DRL (F-DRL), where base stations work together to allocate the optimal transmit power by sharing their models' weights. Our work is different from these previous works in terms of the problem's conception and formulation. In [14] for example, they only addressed the problem of power allocation to base stations, while in [13], the researchers tried to maximize the amount of data that the agent offloads while alleviating its interference with adjacent links.

In this work, however, we focus on power allocation in edge computing to design an effective task-offloading solution for IoT devices. We model the problem of optimizing both energy consumption and task execution delay in IoT devices as a Markov Decision Process (MDP). To address this optimization problem, we employ a reinforcement learning technique and propose an ϵ-greedy Q-learning-based task offloading algorithm. We then leverage federated learning to orchestrate and speed up the training of multiple agents simultaneously. We consider many factors in the process of decision-making, such as the stochastic channel gain between the devices and the edge servers, the random process of task generation, the battery power level of the device, and the available resources in the device. The experimental results demonstrate that the proposed algorithm outperforms central DQN in terms of convergence and performance.

III. Problem formulation

A. System model

We consider a wireless cellular network consisting of several stationary IoT devices and edge servers. In every cell, there is an edge server that receives offloaded tasks from a limited number of predefined stationary IoT devices in its coverage area. IoT devices constantly generate a random number of computation tasks, but they possess limited computational capacity and energy, therefore, offloading tasks to the edge servers can improve the QoS requirements. The batteries in these devices are supposed to discharge constantly with a constant discharge rate. We also suppose that the batteries are charged randomly for a random period of time.

B. Communication model

In our study, we partition the time interval into consecutive epochs of fixed duration. The epoch index is denoted by an integer t, where $0 < t \leq T$, and T represents the maximum number of epochs in the considered time horizon.

We assume that the communication between the IoT devices and the edge servers is established using sub-gigahertz radio frequency technology. This allows for reliable and long-range communication with low power consumption, making it ideal for IoT applications where devices are often battery-powered and require a long lifespan. We denote the communication bandwidth between the IoT device and the edge server by B_e. The set of all the IoT devices in the network is denoted by D $=\{d_1, d_2, d_3, \ldots, d_U\}$.

We assume that the channel condition between the IoT device and the edge server changes over time, and we use the channel gain to characterize this change. The channel gain from a given IoT device to the edge server is represented by G_t and remains constant during the offloading time epoch. We assume that the channel gain at every instant is a value selected from the following set G $=\{g_1, g_2, g_3, \ldots, g_n\}$. We denote the task computation latency by L_c and the task transmission latency by L_t. Moreover, we denote the transmit power allocated by the device by P and the computation power consumption by P_c.

Depending on certain parameters, such as the number of tasks in the queue, the channel gain between the device and the edge server, the percentage of the battery power of the device, and its available resources, each IoT device makes its own decision on task offloading by choosing the transmit power allocated for every task. We suppose that the offloading decision is implicitly inferred from the transmission power level allocated by the device. If the transmission power is set to zero, it indicates that the device is processing the task locally. However, if the transmission power is greater than zero, it suggests that the device is offloading the task to the edge server with the chosen transmission power.

C. Task model

We assume that each IoT device has a queue of stored tasks Q = $\{T_1, T_2, \ldots, T_{Max}\}$ that contains independent computation tasks, each with a different size that requires a different number of CPU cycles to process. At every instant t, the device generates a random number based on a fixed task arrival rate. We attribute to every generated task a unique identifier, a randomly selected size from the set of sizes S = $\{s_1, s_2, \ldots, s_M\}$, a number of required resources, and processing time that are both proportionate to the task's size. Each task is also assigned a random priority between 0 and 1 and a deadline, which is calculated as the product of the maximal deadline and the task's priority with some added random noise. If a task remains in the task queue for longer than its defined deadline, it is considered to have exceeded its allotted time and is therefore canceled.

D. Computation model

1) The Local Mode: In the local computing mode (P= 0), the end device executes the computation task itself. We consider that all the IoT devices require a constant number of CPU cycles to process one bit of the computation task which we note d_{device}, and the power consumption of the device per CPU cycle is denoted by P_d. The computation power consumption of the IoT device required to process 1 bit is given by $d_{device} * P_d$. The total power cost required for the whole computation task at the IoT device at time epoch t is calculated as

$$P_{cd} = d_{device} * P_d * s_t \tag{1}$$

Where s_t is the size of the task.

The CPU frequency of the IoT device (F_d) refers to the clock speed of the device's processor, which indicates the number of cycles the device's CPU can execute in a second. The local computing latency (L_{cd}) is defined as the time it takes to execute the task and is calculated as:

$$L_{cd} = \frac{d_{device} * s_t}{F_d} \tag{2}$$

Therefore, the processing cost for the local computing mode is the combination of the local execution latency and local power consumption, and it is given by:

$$C^t_{loc} = (1 - \beta)P_{cd} + \beta L_{cd} \tag{3}$$

With β being the weight factor of the latency cost.

If the device decides not to offload the task, the only cost incurred is the power consumption for local computation P_{cd} and the local task execution latency L_{cd}, and the transmit power allocated by the IoT device, in this case, is set to zero (P= 0).

2) Offloading to the Edge server: The IoT devices are assumed to use the time division multiple access (TDMA)-method to send their data to the edge server. This means that the IoT devices are given specific time slots to transmit data without the other devices interfering, allowing for efficient use of the channel and reducing collisions. The transmit power of the IoT device is represented by P and the resulting achievable transmission rate (bits per second) is indicated as:

$$R^t_e = B_e * log_2\left(1 + \frac{P * G_t}{\sigma^2}\right) \tag{4}$$

Where:
R^t_e represents the achievable transmission rate between the end device and the edge server (in bits per second)
B_e represents the bandwidth of the channel between the IoT device and the edge server (in Hertz)
P represents the transmit power (in watts)
G_t represents the channel gain
σ^2 represents the variance of additive white Gaussian noise (AWGN)

The number of CPU cycles needed to process one bit of the computation task by the edge server is denoted by d_{edge}, and its power consumption per CPU cycle is denoted by P_e. Therefore, its CPU frequency is denoted by F_e. The corresponding computation power at the edge server is calculated as:

$$P_{ce} = d_{edge} * P_e * s_k \tag{5}$$

And the computation latency can be found as:

$$L_{ce} = \frac{d_{edge} * s_k}{F_e} \tag{6}$$

The time delay that occurs when the task's data is transmitted from the IoT device to the edge server is represented by:

$$L_{te} = \frac{s_k}{R^t_e}$$

And the corresponding IoT device's power consumption is represented by P_{ce}. Finally, the processing cost of edge computing can be expressed as the combination of edge computing delay and power consumption:

$$C^t_{edge} = (1 - \beta)(P_{ce} + P) + \beta(L_{ce} + L_{te}) \tag{7}$$

The problem of task offloading formulated in this work involves finding the best trade-off between energy consumption and latency, which are two contradictory objectives. On the one hand, energy consumption is a critical constraint for IoT devices, which are often powered by batteries that have limited capacity. On the other hand, reducing latency is essential for real-time applications. Offloading computation tasks to more powerful servers like the edge servers can potentially reduce energy consumption, but it also introduces additional communication latency due to data transmission over wireless channels. Therefore, the challenge is for every IoT device to achieve a task allocation policy that can minimize the energy consumption of the device while satisfying the desired latency requirements that ensure the service level agreement (SLA).

The aim is to enable the agents (the IoT devices) to make the best decisions regarding the processing location of the tasks and the transmit power levels, based on their computational capabilities, available energy, resources, and network conditions. This can be achieved by minimizing the combined cost of power consumption and latency described by the following cost function:

$$C^t = C^t_{loc} + C^t_{edge} + \delta^t \tag{8}$$

Whereby δ^t is a penalty function that describes the cost that the agent incurs when it takes an action that violates the constraints of the system. The penalty function is used to penalize the agent when it chooses to process a task locally but the device's remaining resources are not enough or if the battery power is not enough. Similarly, if the agent chooses to offload the task to the edge, but the allocated transmission power surpasses the battery power level, a penalty is incurred. The penalty is weighted using penalty factors, and it is proportional to the severity of the violation of the previously mentioned constraints.

The optimization problem for every agent is formulated as the following:

$$\begin{cases} min \sum_{t=1}^{T} C_t \\ 0 \leq P \leq p_{max} \end{cases} \tag{¶}$$

, whereby C_t is the combined cost of power consumption and latency achieved by the agent at instant t and p_{max} is the maximum transmission power that can be allocated by the IoT device.

We notice that this problem is a mixed integer nonlinear programming (MINLP) problem that involves nonlinear functions and integer variables, making it difficult or even impossible to solve using conventional optimization techniques. Furthermore, optimization techniques take an essential time before delivering optimal configuration, making them unfeasible for real-time, whereby the configurations should be supplied online quickly. Reinforcement learning is an attractive approach to tackle this problem, as it provides a framework for learning to make optimal decisions and adapt to the dynamic environment based on the feedback obtained from the environment.

IV. Federated Deep Reinforcement Learning

We propose a novel approach to solving the problem (¶) using federated deep reinforcement learning. We begin by redefining the problem as an RL setting, where each end device is an agent whose objective is to maximize its own reward while satisfying the constraints of the system. Then, we leverage FL to allow multiple agents to collaboratively learn a global model by sharing their model weights while keeping their private experiences local.

The following section will begin by defining the RL problem in relation to its state space, action space, and corresponding reward function in light of the previous problem (¶). Subsequently, we will describe the Federated Deep Q-Network (FDQN) approach in detail.

A. Reinforcement Learning Formulation

The task offloading problem can be formulated as a Markov decision process (MDP), which can be defined as the tuple $\langle S, A, P, R, \gamma \rangle$, where S is the state space, A is the action space, P is the transition probability function, R is the reward function, and γ is the discount factor.

$S = \{s_1, s_2, ..., s_n\}$ is the set of states where each state S can be defined as the combination of the channel gain between the device and the edge server G_t, the task queue of the device Q_t, the percentage of the device battery, and the percentage of the available computation resources in the device.

Thus, we represent the state S_k as:

$$S_k = (G_k, Q_k, R_k, B_k) \tag{9}$$

A is the set of possible actions, which correspond to the discrete transmission power levels ranging between 0 and $P_{\max}$ as the following: [14]

$$A = \left\{0, \frac{P_{\max}}{M-1}, \frac{2P_{\max}}{M-1}, \ldots, P_{\max}\right\} \tag{10}$$

, whereby M is the number of power levels. All agents have the same action space. $A = 0$ corresponds to the decision to process the task locally, while other values correspond to the levels of transmit power allocated if the agent decides to offload to the edge.

P$(s_{t+1}, C_t | s_t, a_t)$ represents the state transition probability, and it specifies the probability distribution over the next state s_{t+1} with the cost C_t given the current state s_t and action a_t taken by the agent according to the policy π.

Meanwhile, γ is the discount factor, which is a value between 0 and 1 that determines the relative importance of future rewards compared to immediate rewards. A discount factor of 0 means that only immediate rewards are considered, while a discount factor of 1 means that all future rewards are given equal weight.

R(s_t, a_t) represents the reward received from the environment after taking the action a_t on the state s_t by the agent, and it is defined as the negative of the cost.

The long-term expected cost is the following:

$$V(s, \pi) = E_\pi \left[\sum_{t=1}^{T} \gamma_t \cdot C_t \right] \tag{11}$$

, whereby $\gamma_t \in$ [0, 1] represents the discount factor, and E is the statistical conditional expectation with transition probability P, which refers to the expected value of the random variable representing the immediate reward plus the discounted expected value of the future rewards, given a certain state and action.

The agent aims at finding the optimal policy π^* that minimizes the long-term expected accumulated discounted Cost V(s, π) over all states S, while considering the state transition probability and the reward function. The policy π^* is an optimal policy if we have:

$$V(s, \pi^*) \leq V(s, \pi), \forall s \in S \tag{12}$$

The Q-value $Q(s, a)$ (action-value function) estimates the expected total discounted cost obtained from taking a particular action $a_t \in A$ in a given state-action pair and following a specific policy π thereafter. The action-value function $Q(s, a)$ is given by:

$$Q(s, a) = E_\pi[C_{t+1} + \gamma Q_\pi(s_{t+1}, a_{t+1}) | s_t = s, a_t = a] \tag{13}$$

A neural network is used to estimate the Q-value for each action in a given state s_t. The Q-value function is learned through the process of trial and error, as the agent interacts with the environment and receives feedback in the form of rewards. The network is trained using a loss function that minimizes the difference between the predicted Q-values and the actual rewards received. Once the Q-values have been learned, they are stored and used to determine the best action to take in a given state.

The update equation used in deep Q-learning is a variant of the Q-learning update equation that involves updating the Q-value based on a target value that is computed using the current estimate of the Q-value and the estimated value of the next state obtained from the neural network. This is known as the ***target Q-value***. The update equation used in deep Q-learning is given by:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left(C_t + \gamma \cdot minQ(s', a') - Q(s_t, a_t)\right) \tag{14}$$

Where:
Q(s,a) is the Q-value for state s and action a
α is the learning rate ($0 < \alpha \leq 1$)
C_t is the immediate cost obtained by taking action a in state
γ is the discount factor
s' is the next state
a' is the action that maximizes the Q-value for the next state
$minQ(s', a')$ is the minimum Q-value over all actions a' in the next state s'

We use the ϵ-greedy approach to balance the exploration of new actions with exploiting actions that are currently estimated to be the best. The ϵ-greedy approach allows the agent to explore new actions while still exploiting the current best estimate of the optimal action. By gradually decreasing the value of ϵ over time, the agent can transition from exploring more to exploiting more as it becomes more confident in its Q-value estimates.

B. Federated Learning Formulation

The DRL algorithms use Deep Neural Network (DNN) to approximate either the optimal Q-values or the probability of taking an action depending on their type. Combining Federated Learning with DRL allows us to exploit the strengths of each technique. FL speeds up the process of learning and allows the data to be processed on the client devices without transferring it to a central server, while DRL allows agents to learn, almost like humans, autonomously by interacting with a dynamic environment.

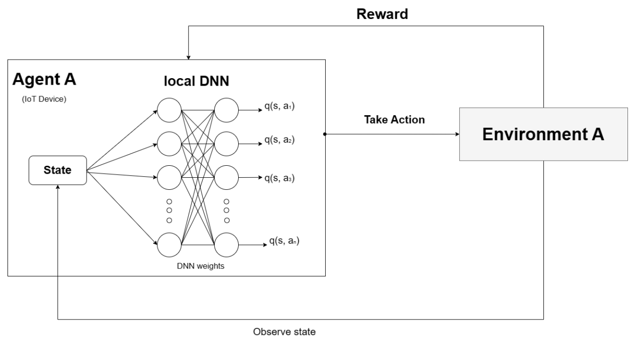

Fig. 1. DQN framework

Federated DRL functions similarly to standard DRL, where an agent interacts with its environment and receives rewards for its actions. However, unlike standard DRL, agents learn in a synchronized way from a global agent to use their decentralized data and exploit the experiences of other agents to update their policies (or its Q-values in the case of value function-based algorithms).

As Fig 2 shows, the agents share knowledge gained from their local information while keeping the information itself decentralized and private. Agents update their local models by training on their local state information and periodically sending their model parameters to the global agent, which will then aggregate them and send the new resulting weights to the local agents. By sharing the updated model weights, the different agents can leverage each other's experiences and improve performance faster without having to share their confidential, bandwidth-intensive data.

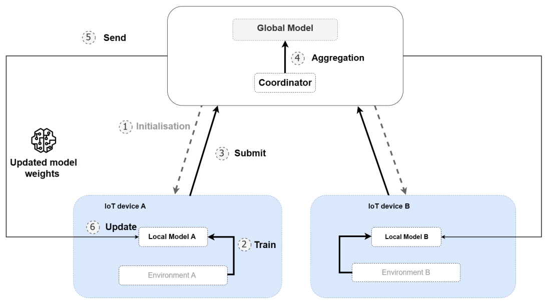

Fig. 2. Federated DQN Architecture

We can formulate the goal of the training as minimizing the following objective function:

$$\min_\theta F(\theta) = \sum_{i=1}^{N} F_i(\theta_i), \tag{15}$$

, whereby $F(\theta)$ and θ represent the global loss function and weights of the global model weights, F_i and θ_i are the local loss function, and the weights of the local model at device i, respectively.

The accuracy of the action-value estimation determines the selection of a good action. Therefore, every DQN model q aims to find the optimal parameters θ^* that minimize the loss function $L(\theta)$ given by: [15]

$$L(\theta_q) = \left(r + \gamma \max_{a'} Q(s', a'; \theta_q^-) - Q(s, a; \theta_q)\right)^2 \tag{16}$$

, whereby θ_q is the current model parameters, θ_q^- is the target network parameters, s is the current state, a is the action taken in the current state, r is the reward received for taking the action, s' is the next state, a' is the next action, and γ is the discount factor.

Similar to classical Q-learning, the agent collects experiences by interacting with the environment. The network trainer constructs a data set D by collecting the experiences until time t in the form of $(s_{t-1}, a_{t-1}, r_t, s_t)$, and they are then used to optimize the loss function $L(\theta_q)$. In the early stages of training, a dynamic ϵ-greedy policy is adopted to control the actions, where the agent with a certain probability explores different actions regardless of their reward. This strategy promotes accurate estimation over time and reduces the risk of over-fitting the model to actions with high rewards in the first phase of training.

By substituting the DQN cost function into the objective function of FDQN, we obtain the FDQN cost as:

$$\min_{\theta_q} L(\theta_q) = \sum_{i=1}^{N} L_i(\theta_{q,i}) \tag{17}$$

V. Simulation and Results

We have conducted a simulation study to evaluate the performance of the proposed Federated Deep Q-Network (FDQN) task offloading scheme in comparison to the central Deep Q-Network (DQN) approach. We implemented both schemes using the PyTorch framework and we trained both models with $10,000$ training episodes, each episode consisting of 20 epochs. We modeled the task generation process using a stochastic approach. To simulate the task arrival process, we set the arrival rate of tasks to 5 tasks per second. We then generated a sequence of inter-arrival times between tasks using a random variable drawn from the Poisson distribution. We set the maximal length of the Task queue to 10.

The initial power of the IoT device is set to $0.1\ W$, the constant discharging rate is set to $5 \cdot 10^{-4}$, and the charging power is set to $0.08\ W$. We considered the resource-constrained nature of IoT devices and modeled it by setting the granularity of battery power to $5\ \%$ and that of the remaining resources to 2%. This means that we discretized the battery power and remaining resources of each device into a finite number of levels, to reflect the practical limitations of such devices. β the weight factor is set to 0.5. We also set the beginning and the end of the charging time randomly. For the transmit power, we set the number of power levels to 10, and the maximal power to $23\ dB$. Other simulation details are resumed in Table I.

TABLE I
Parameters Simulation

S	[5, 10, 15, 20] Kbits
B_e	10^5
σ	-160 + 10*$\log_{10}(10^5)$
β	0.6
Channel Gain	[0.5, 0.6, 0.7, 0.8, 0.9, 1, 1.1, 1.2, 1.3, 1.4, 1.5]
γ, α	0.95, 0.001
d_{device}, d_{edge}	600, 500 cycles/bit
P_d, P_e	$6 \times 10^{-7}, 1 \times 10^{-8}$ W per CPU cycles
F_d, F_e	500MHz, 4GHz

We have measured the running average reward and the convergence time of fully central DQN and the proposed FDQN approach with 4 agents. As shown in Fig 3, the proposed FDQN scheme exhibits a remarkable convergence in the mean reward of the agents over 10,000 training episodes. The average reward is observed to gradually increase, which implies that the FDQN approach is able to effectively learn the optimal policies for task offloading while considering the constraints of the local devices. This is an encouraging result as it shows that the FDQN approach is capable of achieving significant improvement in the overall system performance over time. The convergence in the mean reward also indicates that the agents were able to effectively coordinate their actions to achieve the desired task-offloading objectives.

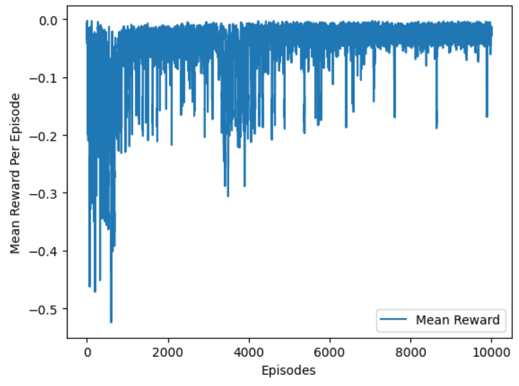

Fig. 3. Convergence performance of the proposed task offloading algorithm measured by the average reward of agents, the weight factor β = 0.5.

The results in Fig 4 show that the proposed FDQN approach outperforms the central DQN approach in both average reward and convergence time. While both models increased in reward over time, the FDQN demonstrated a much smoother and organic increase, with fewer fluctuations and more stable convergence. On the other hand, the centralized DQN exhibited more volatility and instability in all the stages of training, leading to a less steady convergence to the optimal reward. These results indicate the superiority of the federated learning approach over the centralized one, as the former leverages the benefits of distributed learning to achieve better performance and stability over time.

Overall, our simulation results demonstrate the potential of the proposed FDQN approach for speeding up the training process and improving the efficiency of task offloading in IoT networks. It provides a decentralized solution that can adapt to the dynamic nature of IoT environments and provide better performance compared to the traditional central approaches.

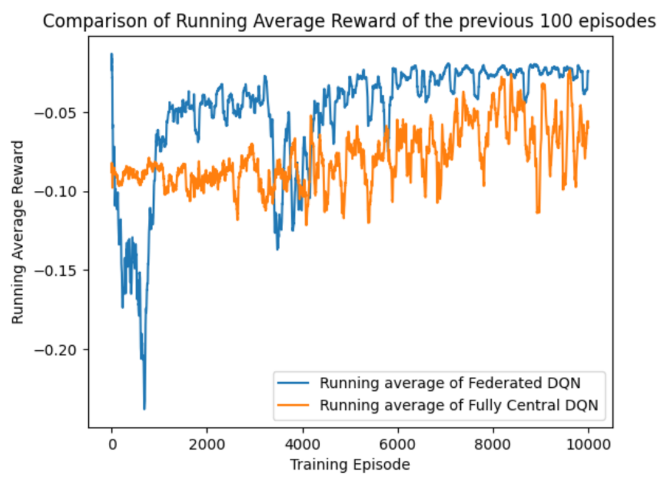

Fig. 4. Comparison of the Running average of the previous 100 episodes between fully central DQN and FDQN, the weight factor $\beta = 0.5$.

VI. Conclusion

In this paper, we have proposed a federated learning-based approach to DQN, in which multiple federated agents collaborate to efficiently address the task offloading problem for IoT devices in the Edge computing environment. They achieve this by sharing the weights of their individual embedded DNNs rather than the data generated by the models. We began by presenting the task offloading and power allocation problem, formulating it as a Federated Deep Reinforcement Learning challenge. The evaluation results of this method regarding performance and convergence demonstrated its superiority when compared to the central approach.

Future research can focus on further enhancing the proposed method for task offloading in various scenarios, including the option of offloading to the cloud and considering mobile IoT devices rather than stationary ones. Moreover, this method can be refined to optimize power consumption for Edge servers. Additionally, it is important to test and compare this method to other existing baselines.

References

[1] S. Chen, S. Sun, H. Chen, J. Ruan, and Z. Wang, "A game theoretic approach to task offloading for multi-data-source tasks in mobile edge computing," in *2021 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*. IEEE, 2021, pp. 776–784.

[2] K.-H. Kim, J. Lynskey, S. Kang, and C. S. Hong, "Prediction based subtask offloading in mobile edge computing," in *Proceedings of the 2019 International Conference on Information Networking (ICOIN)*. IEEE, 2019, pp. 448–452.

[3] T. Zhao, S. Zhou, L. Song, Z. Jiang, X. Guo, and Z. Niu, "Energy-optimal and delay-bounded computation offloading in mobile edge computing with heterogeneous clouds," in *China Commun*, vol. 17, 2020, pp. 191–210.

[4] X. He, H. Xing, Y. Chen, and A. Nallanathan, "Energy-efficient mobile-edge computation offloading for applications with shared data," in *arXiv preprint arXiv:1809.00966*, 2018.

[5] M. Avgeris, D. Spatharakis, D. Dechouniotis, N. Kalatzis, I. Roussaki, and S. Papavassiliou, "Where there is fire there is smoke: a scalable edge computing framework for early fire detection," *Sensors*, vol. 19, no. 3, p. 639, 2019.

[6] Y. Nan, W. Li, W. Bao, F. Delicato, P. Pires, Y. Dou, and A. Zomaya, "Adaptive energy-aware computation offloading for cloud of things systems," *IEEE Access*, vol. 5, pp. 23 947–23 957, 2017.

[7] J. Xu, L. Chen, and P. Zhou, "Joint service caching and task offloading for mobile edge computing in dense networks," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 207–215.

[8] M. Avgeris, D. Dechouniotis, N. Athanasopoulos, and S. Papavassiliou, "Adaptive resource allocation for computation offloading: A control-theoretic approach," *ACM Transactions on Internet Technology (TOIT)*, vol. 19, no. 2, pp. 1–20, 2019.

[9] C. Sonmez, A. Ozgovde, and C. Ersoy, "Fuzzy workload orchestration for edge computing," *IEEE Transactions on Network and Service Management*, 2019.

[10] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji, and M. Bennis, "Performance optimization in mobile-edge computing via deep reinforcement learning," *arXiv preprint arXiv:1804.00514*, 2018.

[11] C. Zhang, Z. Liu, and B. Gu, "A deep reinforcement learning based approach for cost-and energy-aware multi-flow mobile data offloading," *IEICE Trans. on Commun.*, pp. 1625–1634, Jan. 2018.

[12] X. Liu, Z. Qin, and Y. Gao, "Resource allocation for edge computing in iot networks via reinforcement learning," in *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, May 2019, pp. 1–6.

[13] S. Moon and Y. Lim, "Federated deep reinforcement learning based task offloading with power control in vehicular edge computing," *Sensors (Basel)*, vol. 22, no. 24, p. 9595, Dec. 2022.

[14] P. Tehrani, F. Restuccia, and M. Levorato, "Federated deep reinforcement learning for the distributed control of nextg wireless networks," in *2021 IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN)*, Dec. 2021, pp. 248–253.

[15] D. Karunakaran, S. Worrall, and E. Nebot, "Efficient statistical validation with edge cases to evaluate highly automated vehicles," *arXiv preprint arXiv:2003.01886*, 2020.