


RESEARCH

Open Access



A reinforcement learning-based computing offloading and resource allocation scheme in F-RAN

Fan Jiang^{1*} , Rongxin Ma¹, Youjun Gao² and Zesheng Gu¹

*Correspondence:

fjiangwbc@gmail.com

¹ Shaanxi Key Laboratory of Information Communication Network and Security, Xi'an University of Posts and Telecommunications, Xi'an, China

Full list of author information is available at the end of the article

A preliminary version of this paper has appeared in PIMRC'2020 (Fan Jiang et al., 2020). The current version contains the extended Dueling Deep Q-Network Learning-Based Computing Offloading Scheme for F-RAN.

Abstract

This paper investigates a computing offloading policy and the allocation of computational resource for multiple user equipments (UEs) in device-to-device (D2D)-aided fog radio access networks (F-RANs). Concerning the dynamically changing wireless environment where the channel state information (CSI) is difficult to predict and know exactly, we formulate the problem of task offloading and resource optimization as a mixed-integer nonlinear programming problem to maximize the total utility of all UEs. Concerning the non-convex property of the formulated problem, we decouple the original problem into two phases to solve. Firstly, a centralized deep reinforcement learning (DRL) algorithm called dueling deep Q-network (DDQN) is utilized to obtain the most suitable offloading mode for each UE. Particularly, to reduce the complexity of the proposed offloading scheme-based DDQN algorithm, a pre-processing procedure is adopted. Then, a distributed deep Q-network (DQN) algorithm based on the training result of the DDQN algorithm is further proposed to allocate the appropriate computational resource for each UE. Combining these two phases, the optimal offloading policy and resource allocation for each UE are finally achieved. Simulation results demonstrate the performance gains of the proposed scheme compared with other existing baseline schemes.

Keywords: Fog radio access networks, Computing offloading, Resource allocation, Deep reinforcement learning, Dueling deep Q-network, Deep Q-network

1 Introduction

Our society is becoming increasingly highly digitized, hyper-connected and globally data driven. Many widely anticipated services, including virtual reality (VR), augmented reality (AR), internet of vehicles, and ultra-HD video, are all call for extreme-low latency and extreme-low energy consumption in the 6G system [1]. These novel applications usually require powerful computational capacity, huge amounts of energy, and rigorous delay constraints. However, user equipments

(UEs) usually have limited computational capability. Therefore, it is not practical to run such complicated applications on the mobile devices of UEs. To solve this problem, fog radio access network architecture (F-RANs) is proposed as an extension to the

cloud radio access networks (C-RANs), which is deployed much closer to the UEs, and could provide the accessible computational resource on the heterogeneous fog access points (FAPs) [2]. With the help of F-RANs, UEs can offload their intensively computational tasks to the proximate FAPs to process to decrease latency rather than offload those tasks to the remote cloud center, which not only enable the requirement of UEs to be satisfied at anytime and anywhere but also relieve the pressure from the volume data traffic of the fronthaul links. Furthermore, device-to-device (D2D) communication can be deployed in F-RANs to serve as an effective offload method by providing direct communication and computational capabilities between nearby UEs at the edge of the network.

However, due to the limited computational resource on FAPs, how and where to offload the UE's tasks while optimizing the computational resource in FAPs are hot issues in academic research [3]. Particularly, for computational tasks of UEs with different latency or energy requirements, how to strategically make an optimal offloading and computational resource allocation policy to ensure that UEs' quality of service (QoS) as well as to reduce the total cost is an essential task that many valuable works have been carried [4–7]. For instance, Ref. [4] jointly optimizes the offloading policy and radio resource allocation to satisfy the diverse QoS requirements of multi-UEs in the scenario of multiple FAPs by utilizing the genetic algorithm. With the same scenario as considered in Ref. [4], Ref. [5] utilizes separable semi-definite relaxation to minimize the cost of energy and delay for all UEs. Ref. [6] adopts a traffic prediction method to conduct a problem of dynamic traffic offloading and resource allocation to achieve the compromise of latency and energy consumption. In the same way, Ref. [7] achieves a joint optimization of offloading policy and computational resource based on the combination of fog computing and cloud computing. Moreover, to provide more computational resource at the radio access network to enhance the computation capability, the advantages of D2D communication technology have been explored in the scenario of F-RAN in Ref. [8, 9]. Specifically, Ref. [8] studies partial task offloading in a D2D-assisted F-RAN, where tasks are performed by multiple D2D users or FAP, with the objective of maximizing total utility of the F-RAN system, Ref. [8] combines offloading policies, computational resource allocation and the selection of D2D pairs into consideration. Ref. [9] proposes that UEs who need task offloading can choose the D2D execution mode, FAP execution mode, or cloud execution mode, and utilizes a three-layer graph matching algorithm to obtain the selection space supported by the three execution modes, finally achieve the minimum cost of the considered system.

On the other hand, machine learning (ML) methods have been widely incorporated into the 6G wireless networks in recent researches [10]. Specifically, among the ML-based methods, reinforcement learning (RL) algorithm is greatly suitable for solving the problem with dynamically changing Channel State Information (CSI) in wireless networks. For instance, Ref. [11] utilizes the Q-learning algorithm to make offloading decisions by obtaining the biggest reward to minimize the energy consumption in the wireless system. However, although Q-learning can solve the problem of offloading decisions, it still has some issues in practice. For instance, in the Q-learning algorithm, we need to compute and store each state-action value expressed as Q value produced from each interaction with the environment into a Q-table. However, as the number of the

state-action pair increases, the scale of the Q-table becomes too large to manage and enquire, which makes it complicated to obtain the optimal solutions efficiently and thus will directly affect the QoS of UEs. Therefore, a deep reinforcement learning (DRL) algorithm called deep Q-network (DQN) algorithm is proposed in Ref. [12] to obtain the optimal offloading policy and resource allocation, which utilizes the deep neural network to estimate the Q value which is more efficient to solve the problem concerning massive data. Researches such as [13–15] also show that the DQN algorithm can achieve a better performance in dealing with the problem of offloading decision and resource allocation.

However, a potential drawback with the DQN algorithm is that the value function in some states is independent of the selected action. To deal with this dilemma, an advanced DRL algorithm called dueling deep Q-network (DDQN) algorithm is suggested to overcome the mentioned defect [16]. The core idea of the DDQN algorithm lies in that the state-action Q value in the neural network is further divided into two parts, namely, the value function independent of action, and the action advantage function related to action. Moreover, based on the network structure of DDQN, the agent in RL will eventually learn more accurate value, which means the DDQN algorithm could get better performance than the DQN algorithm in solving the problem related to offloading policy and resource. Ref. [17] adopts a DDQN algorithm to predict the offloading behavior of UEs who have tasks with a semi-online distribution, while calculating and updating the total rewards after each offloading decisions until the total rewards achieve maximum. Similarly, Ref. [18] uses the DDQN algorithm to predict UEs' offloading modes meanwhile achieving a load balance of the MEC server with unknown environment information, which is interpreted as the unknown channel state information. By adopting the DDQN algorithm, Ref. [18] effectively improved the offloading efficiency and decreasing the resource costing. Therefore, in our previous work [19], we studied a computing offloading policy for multiple user equipments (UEs) in F-RANs by using the DQN algorithm to optimize the total utility of UEs. However, the limitation of the work is that computational resource for FAPs has not been optimized. So how to design an effective offloading policy as well as an efficient resource allocation scheme is essential to improve the total utility of UEs.

Illuminated by the contributions of the aforementioned researches, this paper serves to find an optimal offloading policy of UEs' tasks while optimizing the computational resource of FAPs in the considered F-RAN, assuming that each UE has a computationally intensive task to be processed. Especially, since the FAPs are resource-limited, some of the tasks can be processed in the FAP, and the others are forwarded to the cloud server by the fronthaul link. Moreover, some idle UEs who can provide additional computational resource by D2D communication around the requested UEs are taken into consideration to enhance the space of the offloading selection. Consequently, tasks of the requested UEs can be offloaded to FAP, nearby idle UE, cloud server, or process locally, respectively. This problem is formulated as a joint optimization to the offloading policy selection and the computational resource allocation of all UE's tasks with the objection of maximizing the total utility of all requested UEs. This problem has also been proved as a non-convex mixed-integer nonlinear programming (MINP) problem in Ref. [5]. To solve this challenging problem, we decompose it into two phases. At the first phase, a

centralized DDQN algorithm is utilized to select the most appropriate offloading mode for each UE. Especially, we utilize a pre-processing procedure to reduce the complexity of the used DDQN algorithm. At the second phase, based on the training results of the DDQN algorithm, the tasks offloaded to the FAPs are classified according to their delay and energy requirements initially. Then, a distributed DQN algorithm is adopted to optimize the computational resource in each FAP to obtain the final offloading policy and resource allocation.

The contributions of this paper can be summarized as below:

1. Offloading policy selection:

A centralized DDQN algorithm is utilized in the proposed scheme to select the most appropriate offloading mode for each UE, which consists of offloading to FAP, nearby idle UE, or processing by itself. Due to the complexity of the centralized algorithm in the Base Station (BS) will increase with the increasing number of UEs who have the offloading requirements, a pre-processing procedure is adopted to reduce the complexity of the DDQN algorithm by directly satisfying some of the UE's task requirements.

2. Optimize the computational resource in FAPs:

In the second step, there exists a circumstance that multiple UEs might be connected to the same FAP for task offloading. Therefore, to ensure the maximization of the total utility, some of the tasks in FAP should be sent to the cloud server to process. Aiming to jointly allocate the computational resource in FAPs while deciding the offloading decision for the UEs whether they be sent to the cloud or stay at FAP to process, we put forward a distributed DQN algorithm, and combining with the training results of the DDQN algorithm, the final optimal offloading policy and resource allocation are obtained.

3. The performance of the proposed DDQN and DQN algorithms:

Simulation experiment compares the proposed offloading policy and resource allocation scheme with other existing baseline schemes. Meanwhile, the performance of the proposed DDQN algorithm and DQN algorithm also is given.

This paper is organized as follows: Sect. 2 describes the system model, computation model and problem formulation. The proposed offloading policy based on the DDQN algorithm, and the computational resource allocation scheme based on the DQN algorithm, are illustrated in Sect. 3. Simulation results are demonstrated in Sect. 4. Finally, conclusions are drawn in Sect. 5.

2 Method

This section introduces the methods used in the work. we first build formulas of the latency, energy consumption in the considered task processing modes, then the formula described the total utility of the required UEs can be derived. Accordingly, a centralized DDQN algorithm is adopted to solve the problem of offloading mode selection, based on the training results of the DDQN algorithm, a distributed DQN algorithm is utilized to optimize the computational resource of each FAP. Both of the above deep reinforcement learning algorithm is implemented with Python 3.7.7, TensorFlow 2.0, and the Adam

optimizer is used to carry out the gradient descent algorithm to minimize the loss function of the two neural network.

3 System and computation model

3.1 System model

The system structure is shown in Fig. 1, a F-RAN architecture is considered which includes a single cell scenario consisting of M FAPs, N UEs, and K distributed computing nodes (DCNs), where DCN typically acts as an idle UE with adequate computational resource distributed around UE, and can be connected with UEs by D2D communication. Assume each UE only has one computationally intensive task to be dealt with which is characterized as t_n , where B_n is the number of CPU cycles required for computing 1-bit data, and D_n is the size of the task data. To improve the gains brought by offloading behavior, we suppose that each UE has four offloading options. Specifically, each UE can offload its complete task either to FAP, nearby DCN, cloud server, or process locally by itself. Accordingly, the vector of the offloading decision made for the UE n is defined as $d_n, \beta_m, \beta_k, \lambda_{Local}, \lambda_{Cloud} = \{0, 1\}, \forall m \in M, \forall k \in K$. Specifically, parameter “1” indicates the full task of UE will be offloaded, while parameter “0” indicates the full task of UE will not be offloaded. Besides, suppose that all the UEs, FAPs, and DCNs have already cached some task results in their own storage according to an optimal caching matrix proposed by our previous work [20]. Hence, UEs could first search the desired result of their requested task before carrying out the offloading behavior. If, fortunately, the result can be found within the caching storage, the requirement of the UEs can be directly satisfied, and there is no need to offload.

For the sake of characterizing the network topology in the considered F-RAN, we build a matrix $P = [p_{i,j}]_{(M+K) \times N}$ where the columns indicate all the UEs who need to carry out offloading and the rows consist of the FAPs and DCNs which can be associated with. If the distance between UE j and FAP j (or DCN j) exceed the maximal distance of FAP or DCN represented as $d_{FAP}, d_{D2D}, p_{i,j} = 0$, otherwise $p_{i,j} = 1$.

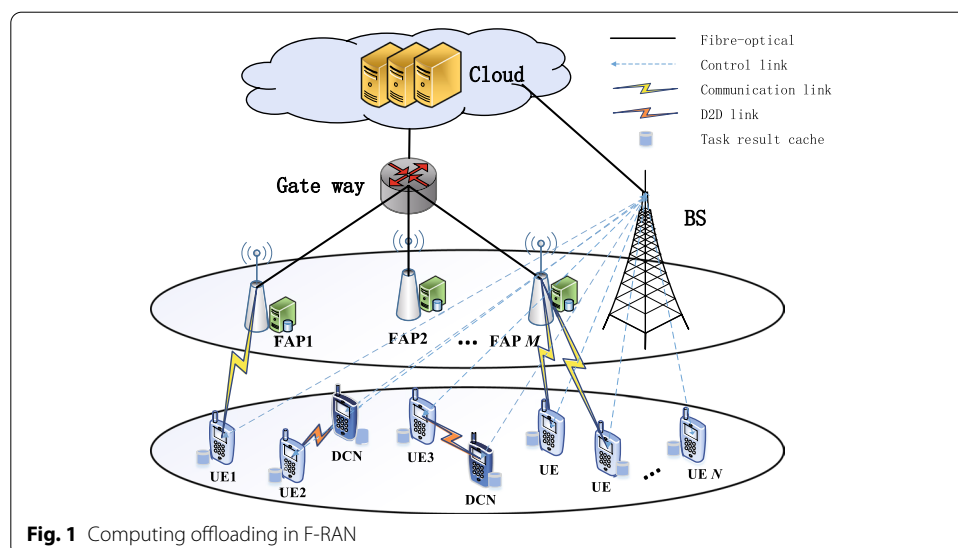


Fig. 1 Computing offloading in F-RAN

Moreover, considering that in practical situation, there might have some DCNs not willing to provide their computational resource, we build a matrix $Y = [y_{ij}]_{K \times N}$ to donate the willingness of DCNs to participate in the offloading process, where y_{ij} is expressed as

$$y_{i,j} = \begin{cases} 0, & \text{the DCN } i \text{ has not willingness to UE } j, i \in K, j \in N \\ 1, & \text{the DCN } i \text{ has willingness to UE } j, i \in K, j \in N \end{cases} \quad (1)$$

Then, matrix Y is used to update matrix P , if $y_{ij} = 1$, then $p_{ij} = 1$, otherwise $p_{ij} = 0$. Therefore, the matrix P can be updated as p_{ij} .

$$p_{i,j} = \begin{cases} 1, & y_{i,j} = 1 \text{ and } d_{i,j} \leq d_{D2D}, i \in K, j \in N \\ 1, & d_{i,j} \leq d_{FAP}, i \in M, j \in N \\ 0, & \text{otherwise} \end{cases} \quad (2)$$

The important parameters in this paper are listed in Table 1.

3.2 Computation model

An orthogonal frequency-division multiple access (OFDMA) method is used to access the FAP through the cellular channel. When a UE has the requirement to

Table 1 Some important parameters

Parameters	Value
M	The number of FAPs
N	The number of UEs
K	The number of DCNs
t_n	The task of UE n
B_n	The number of CPU cycles
D_n	The size of the task data
d_n	Offloading decision vector
P	Network topology matrix
d_{FAP}	The maximal distance of FAP
Y	The willingness matrix of DCN
f_n^l	The computational capacity of UE n
Z_n	The energy consumption in per CPU cycle of UE n
ρ_n^t	The weight factors of latency
ρ_n^e	The weight factors of energy
$f_{n,m}$	The allocated computational resource to UE n in FAP m
f_k	The computational capacity of DCN k
T_c	The round-trip transmission delay
f_n^{Cloud}	The allocated computational resource to UE n at the cloud server
f^{FAP}	The computational resource of FAP
f^{Cloud}	The computational resource of cloud server
C_m	The maximum accessible number of FAP
T	The steps in each training epoch
S	The state matrix
C	The optimal caching matrix
N_m	The number of UEs who offload their tasks to FAP m

communicate with FAP or DCN, the BS will allocate one sub-channel (cellular link or D2D link) to the requested UE. Combined with the four possible options, the corresponding four task processing methods are introduced as follows.

Local processing: If the task of UE n is processed locally, the local execution delay is calculated as

$$T_n^l = \frac{B_n D_n}{f_n^l}, \forall n \in N \tag{3}$$

where f_n^l denotes the computational capacity of UE n which means the CPU cycles per second when the tasking is being processed. Moreover, we adopt the same energy consumption model for local processing as in Ref. [21], which is expressed

$$E_n^l = z_n B_n D_n, \quad \forall n \in N \tag{4}$$

where $z_n = 10^{-27} (f_n^l)^2$ represents the energy consumption in per CPU cycle of UE n .

Moreover, the utility function obtained by UE n with local processing mode is defined as the combination of the saved delay and the saved energy compared when the task of UE n is processed locally. Thereby, the utility obtained by UE n in local processing is expressed as

$$U_n^{local} = \rho_n^t (T_n^t - T_n^l) + \rho_n^e (E_n^e - E_n^l) = 0, \quad \forall n \in N \tag{5}$$

where ρ_n^t and ρ_n^e indicate the weight factors of each task, respectively, and the value of these two parameters ranges from 0 to 1, which satisfies $\rho_n^t + \rho_n^e = 1$ [22]. Moreover, since different tasks have different requirements for delay and energy, if the task has a higher requirement for low latency, ρ_n^t will be bigger, otherwise, ρ_n^e will be bigger.

FAP processing: If the task of UE n is carried out in the FAP processing mode, it takes three steps to complete the offloading process. Firstly, the task is initially uploaded to the associated FAP. Then, the task is performed by utilizing the computational resource provided from the FAP. Finally, the result of the task is returned to the requested UE n . Furthermore, since the transmission rate of the cellular link is relatively high while the task result is comparatively small, the transmission delay for task results is omitted in this paper [23]. Particularly, the uploading delay from UE n to the FAP m is calculated as

$$T_{n,m}^u = \frac{D_n}{r_{n,m}^u}, \quad \forall n \in N, \forall m \in M \tag{6}$$

where $r_{n,m}^u$ denotes the uplink rate of UE n to the connected FAP m of the cellular link, which can be expressed as

$$r_{n,m}^u = B \log \left(1 + \frac{p_{n,m}^u |h_{n,m}^u|^2}{N_0} \right), \quad \forall n \in N, \forall m \in M. \tag{7}$$

In expression (7), $p_{n,m}^u$ denotes the transmit power of UE n , $h_{n,m}^u$ represents the uploading channel gain, B indicates the bandwidth in each sub-channel, and N_0 stands for the noise power in each sub-channel. Thereby, the energy consumption of uploading task t_n is presented as

$$E_{n,m}^u = p_{n,m}^u T_{n,m}^u, \quad \forall n \in N, \forall m \in M. \quad (8)$$

Moreover, the task execution delay to process the task t_n with FAP m can be calculated by

$$T_m^e = \frac{B_n D_n}{f_{n,m}}, \quad \forall n \in N, \forall m \in M \quad (9)$$

where $f_{n,m}$ denotes the allocated computational resource to UE n in FAP m . Additionally, the utility obtained by UEs is only related to the delay and the energy consumption spent on the UEs side. Thereby, for UE n who has offloaded its task to the FAP m , the utility can be represented as

$$U_n^m = \rho_n^t (T_n^l - T_{n,m}^u - T_m^e) + \rho_n^e (E_n^l - E_{n,m}^u), \quad \forall n \in N, \forall m \in M. \quad (10)$$

D2D processing: UE n can establish a D2D link to offload its task to a nearby DCN for processing. Let

$$r_{n,k} = B \log \left(1 + \frac{p_{n,k}^{d2d} |h_{n,k}^{d2d}|^2}{N_0} \right), \quad \forall n \in N, \forall k \in K \quad (11)$$

which stands for the achievable data rate for D2D link between user n and the associated DCN k , where $h_{n,k}^{d2d}$ and $p_{n,k}^{d2d}$ denote the channel power gain and transmit power between UE n and DCN k , respectively. Then, transmission delay and the energy consumption of UE n 's with the D2D link can be presented as

$$T_{n,k} = \frac{D_n}{r_{n,k}}, \quad \forall n \in N, \forall k \in K \quad (12)$$

$$E_{n,k} = p_{n,k}^{d2d} T_{n,k}, \quad \forall n \in N, \forall k \in K \quad (13)$$

the execution delay of DCN k is calculated by

$$T_k^e = \frac{B_n D_n}{f_k}, \quad \forall n \in N, \forall k \in K \quad (14)$$

where f_k indicates the computational capacity of DCN k . Likewise, the utility obtained by UE n in DCN processing can be given as

$$U_n^k = \rho_n^t (T_n^l - T_{n,k} - T_k^e) + \rho_n^e (E_n^l - E_{n,k}), \quad \forall n \in N, \forall k \in K. \quad (15)$$

Cloud processing: Since the computational resource of FAPs is always not adequate for multiple UEs, so some tasks in FAPs should be offloaded to the cloud to ensure the total utility is not affected. With cloud processing mode, it takes the following steps for the offloading. Firstly, the task should be upload to the connected FAP, which be further sent to the cloud through the fronthaul link. Secondly, the task is executed by utilizing the computational resource provided from the cloud server. Finally, the computation result of the task is returned to the requested UE n . Specifically, we use T_c to represent the

round-trip transmission delay in the fronthaul link. Thereby, the execution delay in the cloud processing is expressed as

$$T_{cloud}^e = \frac{B_n D_n}{f_n^{Cloud}}, \quad \forall n \in N \tag{16}$$

where f_n^{Cloud} denotes the allocated computational resource to UE n at the cloud server. With reference to the FAP processing mode, the utility obtained by UE n in cloud processing is calculated as

$$U_n^{Cloud} = \rho_n^t (T_n^l - T_{n,m}^u - T_c - T_{cloud}^e) + \rho_n^e (E_n^l - E_{n,m}^u), \quad \forall n \in N, \forall m \in M \tag{17}$$

3.3 Problem formulation

The objection of this paper is to maximize the total utility obtained by all UEs via finding an optimal offloading policy for each UE and optimizing the computational resource in each FAP. Combining with the system model and communication model, the optimization function can be formulated as

$$p : \max \sum_{n=1}^N \partial_m U_n^m + \beta_k U_n^k + \lambda_{Cloud} U_n^{Cloud} + \lambda_{local} U_n^{local} \tag{18}$$

$$\text{s.t. } C1 : \partial_m, \beta_k, \lambda_{Cloud}, \lambda_{local} = \{0, 1\}, \forall m \in M, \forall k \in K \tag{18a}$$

$$C2 : \partial_m + \beta_k + \lambda_{Cloud} + \lambda_{local} = 1, \forall m \in M, k \in K \tag{18b}$$

$$C3 : \sum_{n=1}^N f_{n,m} \leq f^{FAP}, \forall n \in N, m \in M \tag{18c}$$

$$C4 : \sum_{n=1}^N f_n^{Cloud} \leq f^{Cloud}, \forall n \in N \tag{18d}$$

$$C5 : \sum_i \partial_m^i \leq C_m, \forall i \in N, \forall m \in M. \tag{18e}$$

C1 and C2 constrain that each UE can only select one processing mode. C3 and C4 ensure that the computational resource allocated to the offloaded tasks does not exceed the computational resource of FAP or cloud server presented as f^{FAP} and f^{Cloud} , respectively. C5 states that the number of UEs associated with each FAP should not exceed the maximum accessible number defined as $C_m, m \in M$. Ref. [24] has proved that the problem formulated in (18) is a MINP problem, therefore, it is complicated to find the optimal solution by utilizing traditional optimization algorithms. Besides, the scale of the problem (18) also increases rapidly with the increasing number of UEs, which further increases the complexity of the solution. Consequently, the total utility obtained by UEs will also be directly affected. Based on the above challenges, in the following section, the

problem formulated in (18) will be solved by jointly optimize the offloading policy and computational resource allocation. Specifically, we first divide the original problem into two sub-problems. A centralized DDQN algorithm running at BS is initially adopted to select the most appropriate offloading mode for each UE in the first phase. Especially, a pre-processing stage is introduced to decrease the complexity of the proposed DDQN algorithm. In the second phase, based on the training result of the DDQN algorithm, a distributed DQN algorithm is adopted to optimize the computational resource at each FAP. Combining these two phases, the final optimal offloading policy and resource allocation can be obtained.

4 The proposed computing offloading policy and resource allocation

This section mainly introduces the proposed DDQN algorithm-based computing offloading policy, pre-processing stage, and the DQN algorithm-based computational resource allocation scheme, respectively.

4.1 DDQN algorithm-based computing offloading

In the first phase, the utility obtained by each UE cannot be computed directly in the FAP processing mode because the computational resource in each FAP has not been allocated to UE before making the offloading decisions. Hence, we regard all the computational resource as a whole entirety in each FAP that can be allocated to the requested UE in the first phase, and after making the offloading decisions for each UE, we further optimize the computational resource in each FAP. Specifically, a DDQN algorithm is adopted to find the most appropriate offloading mode for each UE. After choosing the appropriate offloading mode, we will continue to decide which tasks should be sent to the cloud center while allocating optimal the computational resource to the FAPs.

4.1.1 Markov decision process

The DDQN algorithm is based on the DRL algorithm, which as a model-free approach can address complicated system settings by dynamically interacting with an unknown environment without any prior knowledge [25]. Meanwhile, DRL also can handle the potentially large state space problem [26]. In our considered F-RAN system, the problem of making offloading decisions for UEs is formulated as a finite Markov Decision Process (MDP) [27]. In our considered F-RAN system, assuming that the time period is divided into total T steps in each training epoch, and $t = (1, 2, 3, \dots, T)$ indicates each step, the parameter T denotes the number of UEs that need to offload tasks. Combining the considered F-RAN system and the DRL algorithm, the four essentials in RL presented as Agent, Action Space, Environment & State, and Immediate Reward, respectively, in each step t are defined as

Agent: The agent is defined as a learner and a decision-maker in RL. Thereby, in our considered F-RAN system, BS is selected as the agent of the DDQN algorithm.

Environment & State: The environment in RL is defined as the set of all possible states, and the essence of RL is to perform actions to cause the state transfer [28]. Therefore, we set a matrix \mathcal{S} as the state, which has the same shape as the matrix \mathcal{P} , and the value of s_{ij} in the matrix \mathcal{S} should only be 0 or 1, $s_{ij} = 1$ represents the agent selects FAP i (or DCN i) for UE j , otherwise $s_{ij} = 0$. At step $t = 0$ in each training epoch, we initialize

the matrix \mathcal{S} as a total zero matrix, then, the agent executes actions to interact with the environment to trigger the change of matrix \mathcal{S} .

Action Space: The BS make the offloading decision for each requested UE according to the network topology matrix \mathbf{P} , and the optimization object is to find the optimal off-loading mode for each UE. Thereby, in the proposed DDQN algorithm, we use $a_t \in A$ to denote the action in the step t , where $A = \{\partial_1, \partial_2, \dots, \partial_M, \beta_1, \beta_2, \dots, \beta_K\}$.

Immediate Reward: The settings of the reward function always need to be related to the objective function [29]. Accordingly, we set the immediate reward r_t in each step t as two parts: If the constraints in Eq. (18) can be all satisfied, the agent will obtain a positive immediate reward r_t represented as the utility obtained by the t -th UE. Otherwise, the reward obtained by the agent is zero. In addition, there exists another situation that the reward is set to be zero, that is $\exists j \in N, \sum_{i=0}^{M+K+1} p_{ij} = 0$, which means the UE j cannot be connected with any FAP or DCN. Therefore, when the reward is zero, it means the UE should carry out local processing. We define the reward function at step t as

$$r_t = \begin{cases} \text{the utility of } t\text{-UE, if (18a)-(18e) is satisfied} \\ 0, \text{ (18a)-(18e) is not all satisfied or } \exists j \in N, \sum_{i=0}^{M+K+1} p_{ij} = 0 \end{cases} \quad (19)$$

At the end of each training epoch, the accumulated reward is represented as the total utility that the requested UEs.

4.1.2 The pre-processing stage

However, the proposed centralized DDQN algorithm in the BS always has a higher algorithm complexity, which is interpreted as the dimensions of the state space in the DDQN algorithm will increase dramatically as the number of the requested UEs increases, which increases the complexity of the DDQN algorithm while decreasing the efficiency of the network training. Thereby, a pre-processing phase is adopted to decrease the dimensions of the state space to improve the total utility obtained by all UEs. Specifically, assume that each UE, DCN, and FAP has cached some processing results of different tasks based on the optimal caching matrix $C_{(M+K+N) \times N}$ come from our previous research [20]. Combined with our considered F-RAN system, we extend the dimension of the matrix $P_{(M+K) \times N}$ to the same dimension as C and fill in "1" where they are extended. Then, dot multiplies the matrix C and obtains a matrix $P' = P \bullet C$. In this way, each task has its own identity to be distinguished from others in P' . Accordingly, when a UE has a task to be processed, it will first check whether the task result has been cached on its local cache. If the result has not been found locally, the identification of the task will be transmitted to BS, and the BS will search the matrix P' then select the closest route to delivery to the requested UE. If the result can be directly obtained in the pre-processing stage, the maximum utility that UE n can obtain is expressed as

$$U_n = \rho_n^t T_n^l + \rho_n^e E_n^l \quad (20)$$

However, if the result cannot be found in the pre-processing stage, the offloading procedure will be adopted. Since the BS server is equipped with a powerful computing server, the searching and delivery of the task result can be completed so fast that the delay to

transmit can be ignored. Therefore, UEs who no longer need to participate in the task offloading can find their task results during the pre-processing phase. In this way, the complexity of the DDQN algorithm can be decreased. The specific algorithm procedure in the pre-processing phase is shown in Algorithm 1.

Algorithm 1 The pre-processing procedure

Input: matrix P , matrix C , N
Output: The utility of UEs who can obtain the task result from the optimal matrix C , the number of UEs requiring task offloading. $P' = P \bullet C$
for $N = 1$ to $N = N$ **do**
 for $row = 1$ to $row = M + K + N$ in P' **do**
 if $identity ==$ the requested task identity **then**
 $U_n = \rho_n^l T_n^l + \rho_n^e E_n^e$
 $sum(U_n)$
 else
 Put n into an empty list
 $T = len(list)$
 end if
 end for
end for

4.1.3 DDQN algorithm

The DDQN algorithm-based offloading scheme is proposed to select the optimal offloading mode for UEs who need to be offloaded after the state space has been decreased. Specifically, the DDQN algorithm is a typical DRL algorithm that utilizes the deep neural network to approximate the state-action Q value with the aim of maximizing the expected accumulated discounted reward and get the optimal action [30]. The Q function is expressed as formula (20)

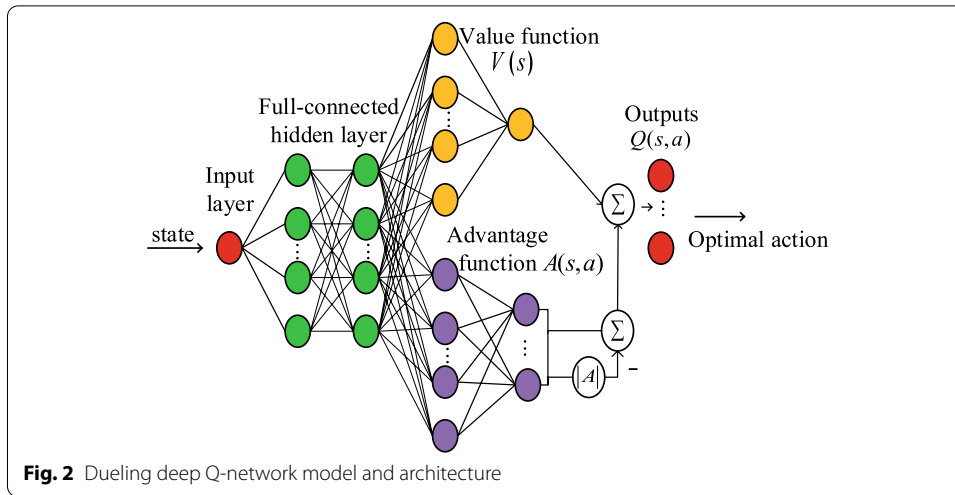
$$Q(s, a) = E \left[\sum_{i=0}^T \gamma^i R_{t+i} | s_t = s, a_t = a \right] \tag{21}$$

where

$$R_t = r_t + r_{t+1} + \dots + r_T. \tag{22}$$

And γ is a discount factor between 0 and 1 that stands for the effect of the future timestamp rewards on current time-step rewards. The greater effect makes a bigger γ .

The model and architecture of the DDQN algorithm we designed is shown in Fig. 2, where we use each step t at a training epoch as an instance to introduce our network model. In each step t , the input of the DDQN network is the current state s_t and the output is the Q value of each possible action at the state s_t , which can be presented as $Q(s_t, a_t)$. The agent selects an action according to the ϵ -Greedy policy then perform the action, which is interpreted as an action is randomly selected with the probability of ϵ and the action that has the maximum value of $Q(s_t, a_t)$ is selected with the probability of $1 - \epsilon$. The advantage of using this ϵ -Greedy policy is that it can make the agent explores the unknown action and state in each step so as to avoid the algorithm falling into a locally optimal solution. After selecting an action to execute, the state will transfer to the next state s_{t+1} . Meanwhile, the agent also gets an immediate reward represented as r_t , and the network will carry on the training at the next step $t + 1$ until the end of the training epoch. During the training process, the object of the DDQN network training is to obtain a series of actions that can achieve the maximized accumulated discounted



reward. This can be interpreted as the BS aims at achieving the maximum total utility for all UEs in the considered F-RAN model. To achieve a better performance of the network training, the DDQN algorithm splits the output $Q(s_t, a_t)$ into two different parts, which is the State Value Function $V(s_t)$ and Action Advantage Function $A(s_t, a_t)$ individually expressed as

$$Q(s_t, a_t, \omega, \varphi) = V(s_t; \omega) + A(s_t, a_t; \varphi) \tag{23}$$

where ω and φ are the network parameters for $V(s_t)$ and $A(s_t, a_t)$, respectively. Specifically, $V(s_t)$ stands for the expected accumulated reward at the state s_t , and $A(s_t, a_t)$ indicates the degree of superiority of action a_t over the average level in state s_t presented as formula (24) and (25).

$$V(s) = E \left[\sum_{i=0}^T \gamma^i R_{t+i} | s_t = s \right] \tag{24}$$

$$A(s_t, a_t) \triangleq Q(s_t, a_t) - V(s_t). \tag{25}$$

According to Ref. [31], formula (23) can reformulated as

$$Q(s_t, a_t; \omega, \varphi) = V(s_t; \omega) + (A(s_t, a_t; \varphi) - \frac{1}{|A|} \sum_a A(s_t, a_t; \varphi)). \tag{26}$$

Furthermore, according to the training procedure of DRL [32], we build the loss function of the DDQN algorithm as

$$L = (r_t + \gamma \max_a \hat{Q}(s_{t+1}, a, \omega^-, \varphi^-) - Q(s_t, a_t; \omega, \varphi))^2 \tag{27}$$

where $r_t + \gamma \max_a \hat{Q}(s_{t+1}, a, \omega^-, \varphi^-)$ represents the target network and $Q(s_t, a_t; \omega, \varphi)$ represents the predict network value. Actually, these two networks have the same structure but different parameters, where the parameters of the former are copied from the latter every I steps. During each training epoch of the DDQN network, the gradient descent

algorithm is utilized to minimize the loss function to find the optimal parameters of the predict network, which is further used to evaluate the Q value of each chosen action [33]. In the DDQN algorithm, an experience pool is introduced to ensure the stability of the network training, where the specific approach is to put the latest interaction data (s_t, a_t, r_t, s_{t+1}) into an experience memory pool, when the training is start, a mini-batch $(s'_t, a'_t, r'_t, s'_{t+1})$ will be randomly sampled from the pool. As a result, the experience replay mechanism not only makes the agent learn from the previous experiences repeatedly but also removes the correlations between the observations. Thereby, the DDQN network training will become more stable and more efficient. The whole procedure of the above proposed DDQN algorithm is drawn in Fig. 3, and the proposed DDQN algorithm-based offloading scheme is presented in Algorithm 2.

Algorithm 2 The proposed DDQN algorithm based offloading scheme.

Input: The number of UEs requiring task offloading T
Output: The utility obtained by the UEs who participate in the task offloading, the optimal offloading decision for each UE, the optimal parameter ω^* and φ^* of the DDQN network
Initialize: The experience memory D and its capacity V , the parameters $\omega^- = \omega$ and $\varphi^- = \varphi$ of the DDQN network, C_m

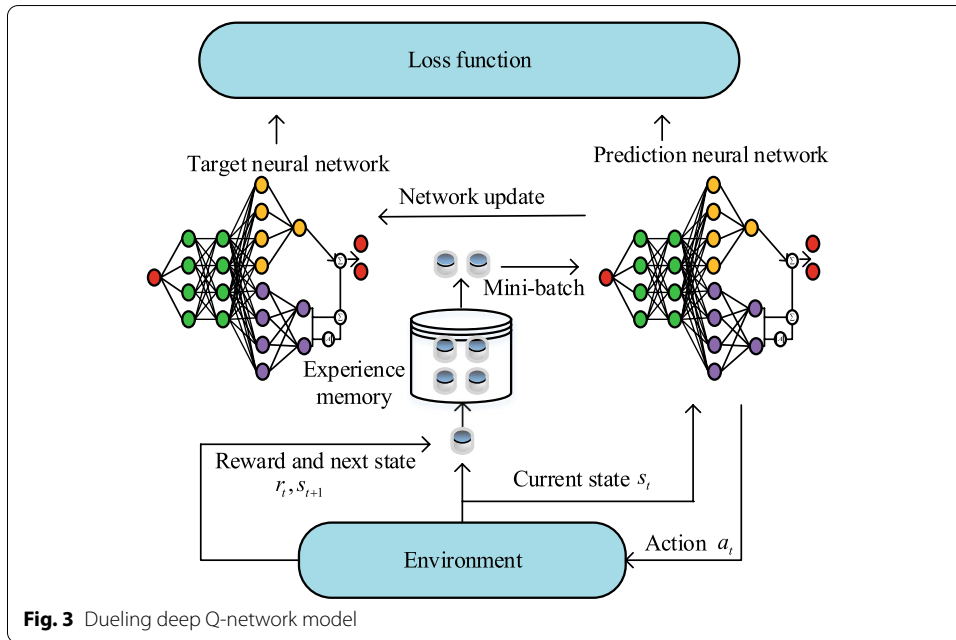
```

for 1, 2, ..., epochs do
  Initialize: The state matrix  $S$  as an all-zero matrix
  for  $t = 0$  to  $t = T$  do
    Select an action  $a_t$  by using  $\epsilon - Greedy$  policy
    Execute  $a_t$ 
    Obtain reward  $r_t$ , next state  $s_{t+1}$ 
    Store the transition data  $(s_t, a_t, r_t, s_{t+1})$  into  $D$ 
     $s_t = s_{t+1}$ 
    if  $V > 50$  then
      Randomly sample a mini-batch transitions data  $(s'_t, a'_t, r'_t, s'_{t+1})$  from  $D$ 
       $Q(s_t, a_t; \omega, \varphi) = V(s_t; \omega) + (A(s_t, a_t; \varphi) - \frac{1}{|A|} \sum_a A(s_t, a; \varphi))$ 
       $L = (r_t + \gamma \max_a \hat{Q}(s_{t+1}, a, \omega^-, \varphi^-) - Q(s_t, a_t; \omega, \varphi))^2$ 
      Reset  $\hat{Q} = Q$  every  $I$  steps
    end if
  end for
end for

```

4.2 DQN algorithm-based computation resource allocation scheme

Since multiple UEs connected to the same FAP will cause resource competition, some of the tasks in FAP should be relayed to the cloud server to ensure the maximization of total utility. Meanwhile, the computational resource in each FAP should be allocated to the UEs whose task has offloaded to the corresponded FAP. In this part, we first classify the tasks in each FAP into two different parts according to UE's different requirements in latency which is characterized with the delay revenue coefficient ρ_n^t . Specifically, tasks with higher delay requirement that are represented as $\rho_n^t \geq 0.5$ are set to be remain at FAP to process. Otherwise, when $\rho_n^t < 0.5$, the tasks will be sent to the cloud to process. Since the cloud center has abundant computational resources and owns powerful processing capability, while the computational resource of FAPs is limited. Thereby, we assume that the tasks sent to the cloud center can be processed in parallel [34]. Meanwhile, a distributed DQN algorithm is adopted to optimize the resource allocation in each FAP.



DQN algorithm is also a typical model-free DQL [35], so the computational resource allocation problem can be formulated as MDP as well, the Agent, State, Action, and Reward are described as follows.

Agent: In the proposed distributed DQN algorithm, since the object is to optimize the computational resource in each FAP, we define the Agent as each FAP.

Environment & State: The state is defined as a combination of the available resources in each FAP and the obtained utility of UE in each FAP, which can be expressed as $s = (F_m, \sum_{i=1}^{N_m} U_i)$, where N_m stands for the number of UEs who offload their tasks to FAP m .

Action Space: The action should contain all possible schemes of resource allocation to the UEs who remain at the FAP m . Besides, the DQN algorithm is mainly oriented to the problem with discrete actions. Thereby, the computational resource in each FAP should also be discrete, and the discrete computational resource blocks should be allocated to each UE. Supposed the computational resource in FAP m is divided equally into X parts. Therefore, the action is expressed as $a_t = (f_1, f_2, \dots, f_i, \dots, f_{N_m}), f_i \in \{1, 2, 3, \dots, X\}$, where f_i denotes the number of computational resource block which is allocated to the UE i .

Immediate Reward: Since the agent act as each FAP in this distributed DQN-based resource allocation problem, so FAP m will immediately get a positive reward denoted as the utility of UEs in FAP m , which is expressed as $\sum_i^{N_m} U_i$. In practice, if the variable range of reward value does not exceed a threshold quantity which is represented as a small value in ten consecutive time steps in the training epoch, we set this training epoch is terminated, and the network will be start at the next training epoch.

Algorithm 3 The proposed DQN algorithm based computational resource allocation.

Input: The available resource of each FAP, $s = (F_m, \sum_{i=1}^{N_m} U_i)$

Output: The optimal parameter θ of the DQN network, the optimal computational resource allocation vector $(f_1^*, f_2^*, \dots, f_i^*, \dots, f_{N_m}^*)$

Initialize: The experience memory D and its capacity V , the parameter $\theta' = \theta$ of the DQN network, the number of resource blocks X , threshold quantity ϑ

```

for FAP 1, 2, ..., M do
  compute  $F_m/X$ 
  for epoch 1, 2, ... do
    for  $t = 0, 1, 2, \dots$  do
      Select an action  $a_t$  by using  $\epsilon - Greedy$  policy
      Execute  $a_t$ 
      Obtain reward  $r_t$ , next state  $s_{t+1}$ 
      Store the transition data  $(s_t, a_t, r_t, s_{t+1})$  into  $D$ 
       $s_t = s_{t+1}$ 
      if  $r_{t+1} - r_t \leq \vartheta$  then
        break
      else
        if  $V > 50$  then
          Randomly sample a mini-batch transitions data  $(s_t', a_t', r_t', s_{t+1}')$  from  $D$ 
          
$$L_t(\theta) = E \left[ \left( \underbrace{\left( r_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}, \theta') \right)}_{\text{Target}} - Q(s_t, a_t, \theta) \right)^2 \right]$$

          Execute the gradient descend algorithm with  $\theta$ 
          Reset  $\hat{Q} = Q$  every  $I$  steps
        end if
      end if
    end for
  end for
end for
end for

```

As shown in Fig. 4, the input of the DQN is the state s_t in each step t , then three fully connected layers are utilized to extract the features of the input data, finally, the output of the DQN is the resource allocation vector. When the DQN algorithm tends to converge, the agent can eventually learn the optimal resource allocation vector $(f_1^*, f_2^*, \dots, f_i^*, \dots, f_{N_m}^*)$.

Similarly, the DQN algorithm uses the gradient descent algorithm to update the Q-network during each training epoch to minimize the loss function, which is formulated as

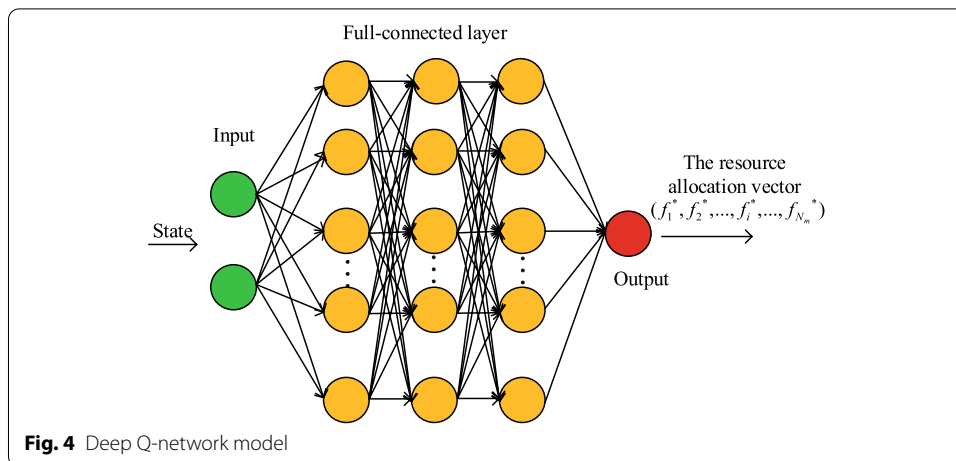
$$L_t(\theta) = E \left[\left(\underbrace{\left(r_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}, \theta') \right)}_{\text{Target}} - Q(s_t, a_t, \theta) \right)^2 \right] \tag{28}$$

where θ' represents the parameter of the target network which is copied from the predict network parameter θ every several steps. As with the DDQN algorithm, the DQN algorithm also adopts the experience replay mechanism to remove the correlation of the data to make the training of the network more stable. The proposed DQN algorithm-based computational resource allocation is illustrated in Algorithm 3.

5 Results and discussion

In this section, the parameters and results of the simulation experiment are presented to verify the performance of our proposed offloading and resource allocation scheme.

We consider a single cell with a radius of 400 m which distributed with 20–100 UEs and 2–10 FAPs. Also, some important parameters are listed in Table 2. Moreover, In



the DDQN algorithm-based offloading scheme, we set the input layer has $(M + K) \times N$ neurons, where we use three fully connected hidden layers to extract the feature of the input data, each of which comprises 256 neurons. Especially, to divide the State Value Function and the Action Advantage Function, we split the third hidden layer into two halves, which means that 128 neurons represent the State Value Function and another 128 neurons represent the Action Advantage Function. Besides, the discount factor γ in the DDQN and DQN algorithm is set as 0.99. Similarly, the DQN algorithm also adopts three full connected hidden layers and each layer comprises 256 neurons. The DDQN and DQN algorithms are implemented by TensorFlow 2.0 based on Python 3.7.7. Moreover, to train the DDQN and the DQN network, we use Adam optimizer and set the learning rate as 0.0002 to realize the gradient descent algorithm to minimize the loss function. For the action selection of each step, we set the parameter ϵ in the ϵ -Greedy policy as decaying from 0.08 to 0.01 through the network training process, which means that the predict Q-network tends to select the action with the maximal Q value to

Table 2 Simulation parameters

Parameters	Values
d_{FAP}	150 m
d_{D2D}	30 m
B_n	Select from [500, 1000] cycles/bit
D_n	Randomly for each UE but fix the average size to 1 MB
f_n	900 MHz
f_{DCN}	1 GHz
f^{FAP}	4 GHz
f_{cloud}	10 GHz
ρ_n^t, ρ_n^e	Uniform distributed in [0, 1]
B	10MHz
N_0	- 174 dBm/Hz
Transmit power of UE	30 dBm
Transmit power of FAP	33 dBm
Channel gains of cellular and D2D links	$\mathcal{CN}(0, 1)$

further improve the learning efficiency. Furthermore, when dividing the computational resources of each FAP, in order to avoid missing the optimal solution, we set that each resource block is 0.1

Figures 5 and 6 display the learning curves representing the accumulated reward obtained by the agent in each training epoch of the DDQN algorithm and the DQN algorithm, respectively. The number of UEs is set as 20, and the average required number of CPU cycles is 760 cycles/bit. We can see that the accumulated reward is gradually increasing with a bit of fluctuation and eventually become stable and converges both in the DDQN algorithm and the DQN algorithm, which indicates that the two networks have been trained perfectly while the optimal offloading decision and resource allocation has finally reached.

Then, we compare the performance of the proposed DDQN algorithm-based offloading scheme with other methods, which are random task offloading scheme (RO), non-D2D offloading scheme (ND2D), non-pre-processing offloading scheme (NP), and random tasks selection scheme of FAP and the cloud (RS), respectively.

Figure 7 shows the total utility of UEs versus the different numbers of UEs. Assume that the average required number of CPU cycles is fixed to 760 cycles/bit. It can be seen from Fig. 7 that the proposed DDQN algorithm-based offloading scheme can achieve the maximal total utility compared with other schemes. We give the following explanations. Firstly, for the RS scheme, since the tasks are randomly selected to be processed by the cloud, thereby, this scheme cannot guarantee the higher requirement of some UEs' tasks for latency. Hence, even the resources have been optimized in each FAP, the utility obtained by some UEs cannot be maximized. Besides, as for the non-pre-processing offloading scheme represented as NP, it can be observed that as the number of UEs increases, the interval between the NP scheme and the proposed DDQN scheme has gradually become larger. This phenomenon can be interpreted as that more UEs indicates more task requirements are produced. Thereby, the probability that the task results could be found within the matrix also increases, resulting in

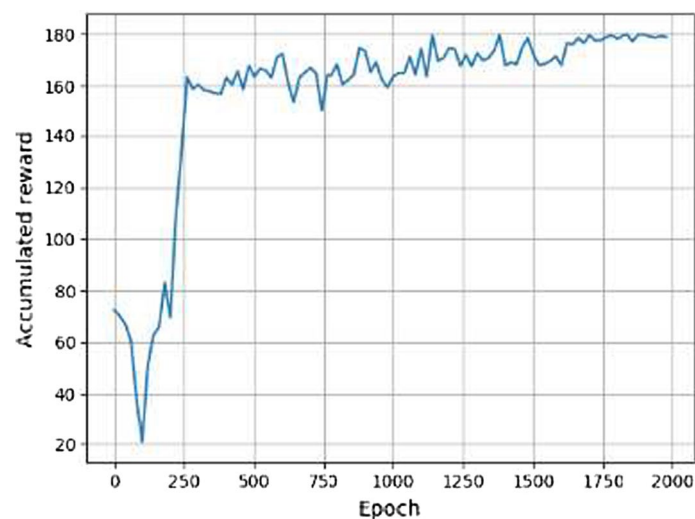
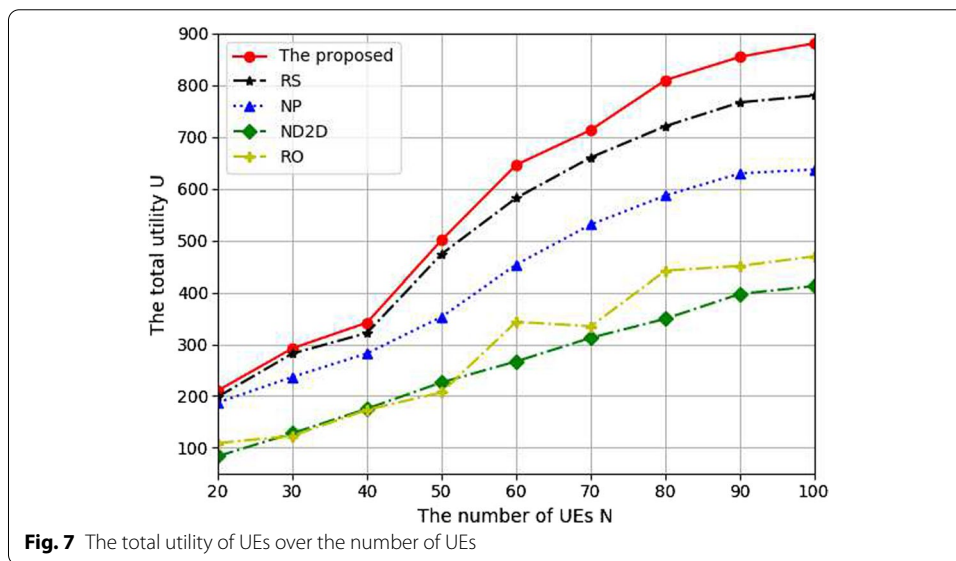
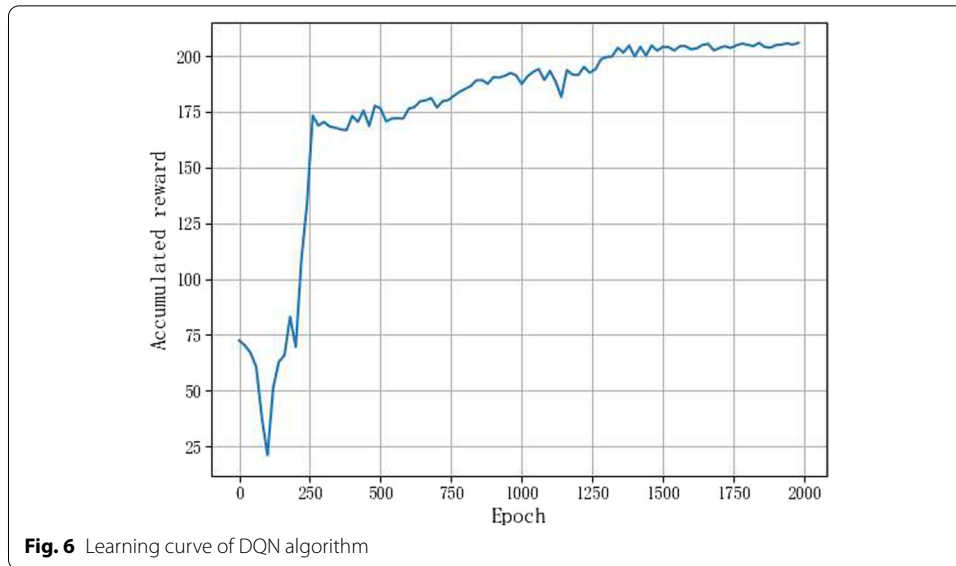


Fig. 5 Learning curve of DDQN algorithm

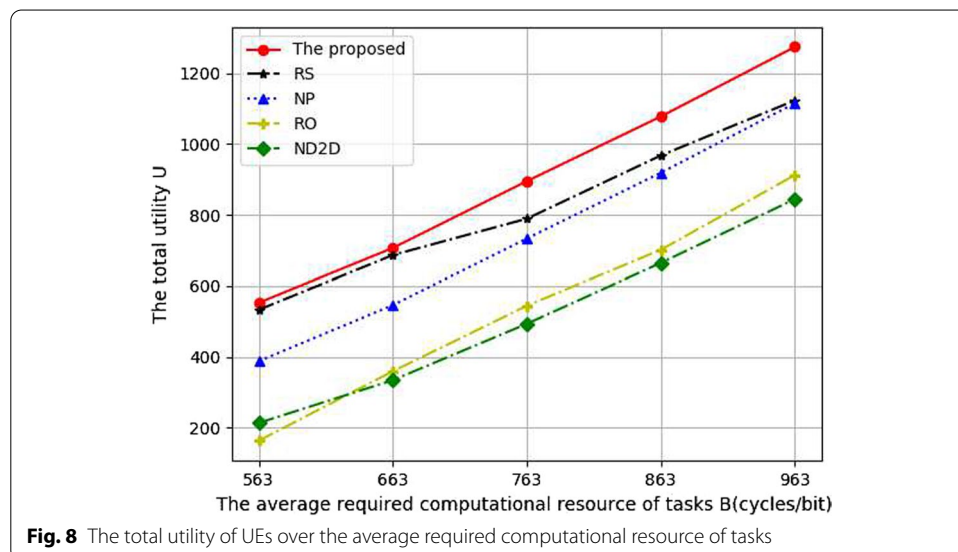


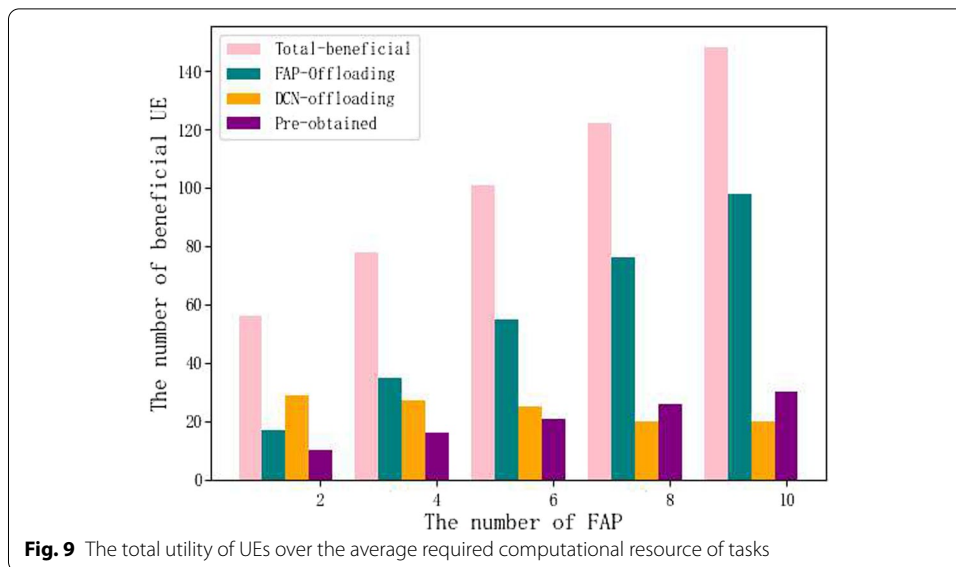
that more UEs can achieve the desired utility by directly obtain the task result which contributes to the gradually increased total utility. Moreover, for the random task off-loading scheme represented as RO, it is clearly shown that the performance is significantly worse than the proposed scheme, the reason might be explained as because the BS selects the farther FAP or proximity devices for some UEs, which results in the increased transmission delay, then the obtained total utility of UEs cannot be satisfied. In some situations, the total utility might even be numerically negative, which is responsible for the poor performance of the total utility. Additionally, for the non-D2D offloading scheme represented as ND2D, the total utility is obviously lower than other schemes. The reason is that without the assistance of the nearby DCNs, tasks of UEs can only be offloaded to FAPs or cloud, and the computational resources of some

nearby DCNs are not well utilized. Thereby, UEs cannot be beneficial from the DCNs, which causes the lowest total utility compared with other schemes.

Figure 8 illustrates the total utility obtained by UEs versus the different average required computational resource of tasks, where we fix the number of UEs to 100. It can be seen that as the average required computational resource of tasks increases, the total utility obtained by UEs also increases. This trend is explained as more required computational resource from UEs means larger execution delays with the local computing method. Accordingly, UEs can benefit by offloading their tasks to FAP, nearby DCNs, or cloud, respectively, through an optimal offloading scheme, which brings larger total utilities than execute the task locally.

Figure 9 demonstrates the number of the beneficial UEs versus the different numbers of FAPs, where we fixed the total number of UE to 150, and the average required number of CPU cycles to 760 cycles/bit. It can be obviously observed that as the number of FAP increases, the total number of UE who can obtain the utility through offloading tasks to others (Total beneficial) also increases. The reason is given as follows. Since more FAPs will provide more alternative offloading modes to more UEs, and consequently UEs can enjoy the more abundant computational resource provided by the FAPs. Compared with DCN offloading mode (DCN offloading), we can observe that as the number of FAPs increase, the number of UEs offloaded to DCNs gradually decreases. The reason behind this phenomenon is twofold. Initially, the competition for offloading opportunities among UEs is fierce when the resource of FAPs is scarce. However, if UE cannot connect to any FAP, it tends to offload to the nearby DCN to increase the utility. Consequently, some tasks of UEs have to be offloaded to the nearby DCN. However, compared with offload the task to DCN, as the numbers of FAPs increase, the computational resource provided by FAPs gradually become abundant to satisfy UEs' demands for lower execution delay of the task. Since the utility obtained from the FAP is larger than that from DCN, more UEs will choose FAP offloading mode for higher utility. Therefore, as the number of FAP increases, more UE will increasingly prefer FAP offloading mode (FAP



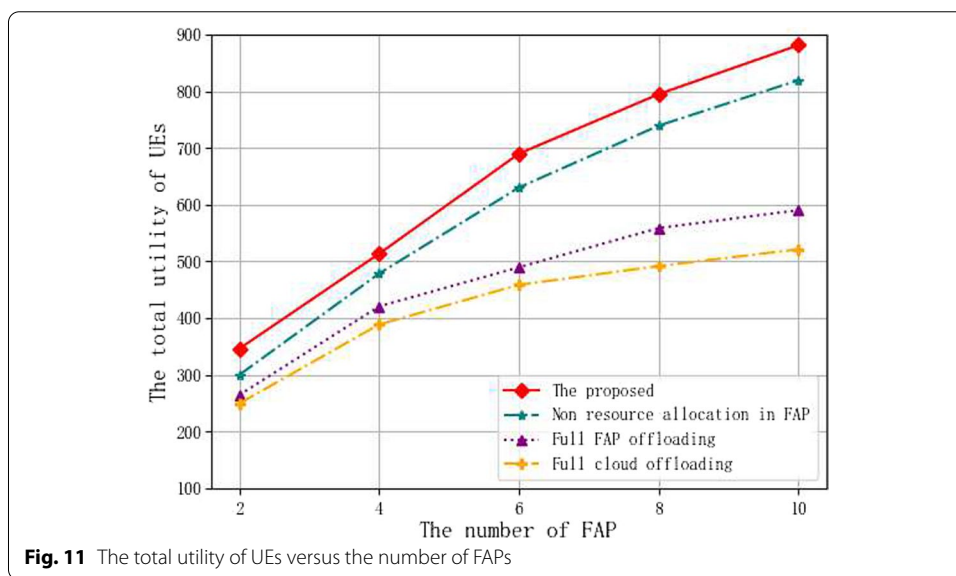
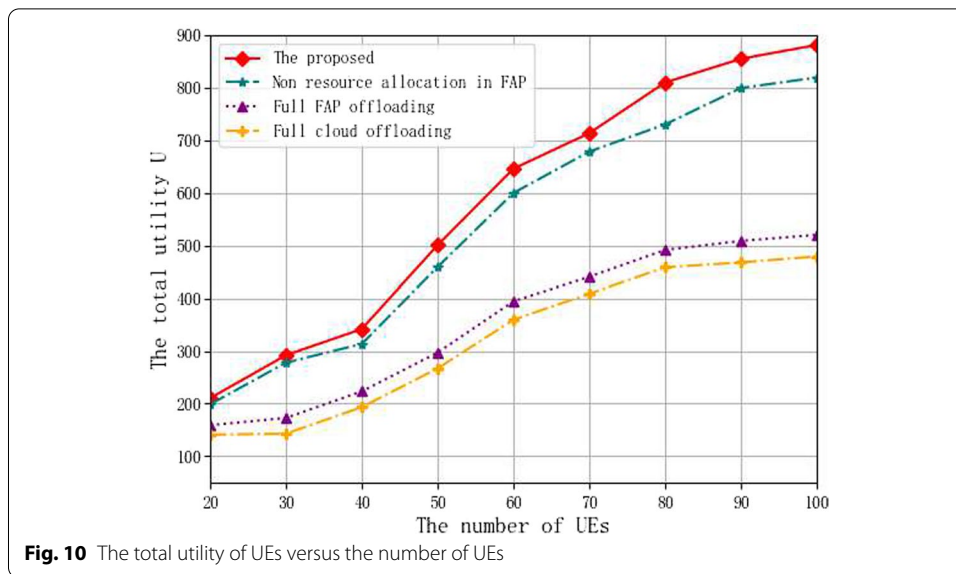


offloading) instead of DCN offloading mode. On the other hand, for UEs who can obtain utility directly during the pre-processing phase (pre-obtained), the increased number of FAP means the larger probability that the task result has been cached, which can satisfy the requirements of more UEs. Therefore, the number of UEs who can directly obtaining the utility in the pre-processing stage grows with the increase in FAP number.

Then, we compare the proposed algorithm with three other methods, namely, Full-FAP offloading where the tasks in FAP remain in FAP to process with non-resource optimization, and Full-cloud offloading which means that the tasks in FAP are all sent to the cloud to process.

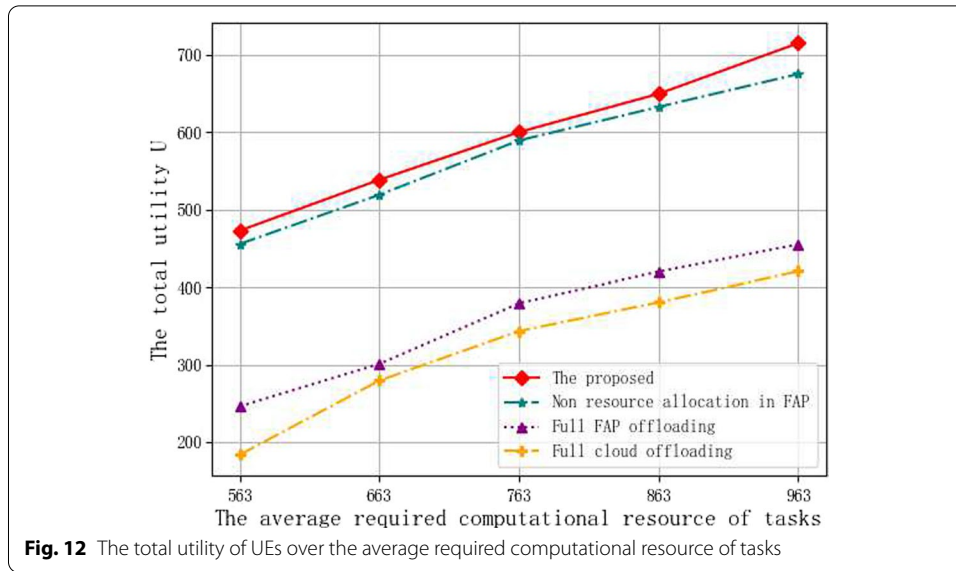
Figure 10 displays the total utility of UEs versus the different numbers of UEs, where we fixed the number of FAP to 10. It can be seen that as the increasing number of UEs, the total utility initially increases then gradually becomes stable. This is because as the number of UEs increases, more computational resource in FAP and cloud need to be allocated to more users, which will incur the longer execution delay of each UE's task. This is responsible for the slow growth of the total utility. Compared to the proposed resource optimization scheme to the non-resource optimization scheme, the DQN algorithm optimizes the allocation of the computational resource in each FAP, which improves the total utility. Besides, the performance of the Full-FAP offloading and the Full-cloud offloading is not as good as expected. This is due to the fact that if all UEs' tasks oriented to the FAP are executed at the FAP, the computational resource of each UE is insufficient, which will directly affect the execution delay. The lower utility of the Full-cloud offloading may be due to the long round-trip delay or the congestion in the fronthaul link, which increases the transmission delay of the task, and affects the total utility.

Figure 11 presents the total utility of UEs versus the different numbers of FAP. It can be seen that the total utility increases with the increasing number of FAP. This is because more FAPs can provide more computational resource, so that more UE with good channel conditions can choose nearby FAP to offload their tasks, which reduces



the transmission delay and execution delay of UEs' tasks, thus improves the total utility of UEs.

Figure 12 illustrates the total utility of UEs versus the average computational resource required of tasks. The quantity of UE and FAP are set to 100 and 10, respectively. It can be observed that the total utility is gradually increasing. This is explained as the more average required computational resource of tasks responsible for a longer execution delay. Compared to processing locally, all offloading schemes such as offloading to FAP, DCN, or cloud will impact the utility of UEs because of the increased delay.



6 Conclusion

In this paper, we have studied an offloading selection and computational resource allocation scheme in F-RAN. Aiming at maximizing the total utility of all UEs who have the task to be processed, a DDQN algorithm-based offloading selection scheme is proposed to initially make the optimal offloading decision for each UE with an unpredictable CSI. Especially, the proposed DDQN algorithm is a centralized algorithm carried out at the BS, so we utilize a pre-processing phase to decrease the complexity of the DDQN algorithm before the network training. After getting the optimal action for each UE, we then utilize the distributed DQN algorithm to optimize the computational resource at each FAP. Simulation results demonstrated that the proposed offloading and resource optimization scheme can effectively increase the utility obtain by the required UEs while achieving a better performance compared with other schemes.

Abbreviations

VR: Virtual Reality; AR: Augmented Reality; UEs: User Equipments; F-RANs: Fog Radio Access Network Architecture; C-RANs: Cloud Radio Access Networks; FAPs: Fog Access Points; D2D: Device-to-Device; QoS: Quality of Service; ML: Machine Learning; RL: Reinforcement Learning; CSI: Channel State Information; DRL: Deep Reinforcement Learning; DQN: Deep Q-Network; DDQN: Dueling Deep Q-Network; MINP: Mixed-Integer Nonlinear Programming; BS: Base Station; DCNs: Distributed Computing Nodes; OFDMA: Orthogonal Frequency-Division Multiple Access; MDP: Markov Decision Process; RO: Random Task Offloading Scheme; ND2D: Non-D2D Offloading Scheme; NP: Non-pre-processing Offloading Scheme; RS: Random Tasks Selection Scheme of FAP and the Cloud.

Acknowledgements

The authors would like to acknowledge all the participants for their contributions to this research study.

Author's contributions

Both authors have contributed toward this work as well as in compilation of this manuscript. The author(s) read and approved the final manuscript.

Funding

This research was supported by the National Natural Science Foundation of China (Grant Nos. 62071377, 61801382, 61901367); The Key Project of Natural Science Foundation of Shaanxi Province (Grant Nos. 2020JQ-849, 2021JM-465, 2019ZDLGY07-06).

Availability of data and materials

Data sharing is not applicable to this article as no datasets were generated or analyzed during the current study.

Declarations

Ethics approval and consent to participate

Not applicable.

Consent for publication

The manuscript does not contain any individual person's data in any form (including individual details, images, or videos) and therefore the consent to publish is not applicable to this article.

Competing interests

The authors declare that they have no competing interests.

Author details

¹Shaanxi Key Laboratory of Information Communication Network and Security, Xi'an University of Posts and Telecommunications, Xi'an, China. ²China Mobile System Integration Co., Ltd., Beijing, China.

Received: 7 May 2021 Accepted: 22 September 2021

Published online: 02 October 2021

References

1. M. Latva-Aho, K. Leppänen, Key drivers and research challenges for 6G ubiquitous wireless intelligence (white paper). Oulu, Finland: 6G Flagship, (2019)
2. Y. Lan, X. Wang, D. Wang, Z. Liu et al., Task caching, offloading, and resource allocation in D2D-aided fog computing networks. *IEEE Access* **7**, 104876–104891 (2019)
3. M. Yang, H. Zhu, H. Wang, Y. Koucheryavy, K. Samouylov, H. Qian, An online learning approach to computation offloading in dynamic fog networks. *IEEE Internet Things J* (2020). <https://doi.org/10.1109/JIOT.2020.3015522>
4. Y. Ma, H. Wang, J. Xiong, J. Diao, D. Ma, Joint allocation on communication and computing resources for fog radio access networks. *IEEE Access* **8**, 108310–108323 (2020). <https://doi.org/10.1109/ACCESS.2020.3000832>
5. M. Chen, B. Liang, M. Dong, Multi-user multi-task offloading and resource allocation in mobile cloud systems. *IEEE Trans Wirel Commun* **17**(10), 6790–6805 (2018). <https://doi.org/10.1109/TWC.2018.2864559>
6. Q. Li, J. Zhao, Y. Gong et al., Energy-efficient computation offloading and resource allocation in fog computing for internet of everything. *China Commun* **16**(3), 32–41 (2019)
7. Y. Lan, X. Wang, D. Wang et al., Task caching, offloading, and resource allocation in D2D-aided fog computing networks. *IEEE Access* **7**, 104876–104891 (2019)
8. X. Chen, J. Zhang, When D2D meets cloud: hybrid mobile task offloading in fog computing, in *2017 IEEE International Conference on Communications (ICC)*. IEEE, 1–6 (2017)
9. A. Bozorgchenani, D. Tarchi, G.E. Corazza, Mobile edge computing partial offloading techniques for mobile urban scenarios, in *IEEE Global Communications Conference (GLOBECOM)*, IEEE, (2018), pp. 1–6
10. N. Docomo, White paper 5g evolution and 6g. Accessed on 1(2020)
11. F. Jiang, W. Liu, J. Wang, X. Liu, Q-learning based task offloading and resource allocation scheme for internet of vehicles, in *2020 IEEE/CIC International Conference on Communications in China (ICCC)*, Chongqing, China, (2020), pp. 460–465, <https://doi.org/10.1109/ICCC49849.2020.9238925>
12. H. Ke, J. Wang, H. Wang, Y. Ge, Joint optimization of data offloading and resource allocation with renewable energy aware for IoT devices: a deep reinforcement learning approach. *IEEE Access* **7**, 179349–179363 (2019). <https://doi.org/10.1109/ACCESS.2019.2959348>
13. S. Nath, Y. Li, J. Wu, P. Fan, Multi-user multi-channel computation offloading and resource allocation for mobile edge computing, in *ICC 2020—2020 IEEE International Conference on Communications (ICC)*, Dublin, Ireland, (2020), pp. 1–6, <https://doi.org/10.1109/ICC40277.2020.9149124>
14. X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji, M. Bennis, Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning. *IEEE Internet Things J* **6**(3), 4005–4018 (2019). <https://doi.org/10.1109/JIOT.2018.2876279>
15. J. Baek, G. Kaddoum, Heterogeneous task offloading and resource allocations via deep recurrent reinforcement learning in partial observable multi-fog networks. *IEEE Internet Things J* (2020). <https://doi.org/10.1109/JIOT.2020.3009540>
16. Z. Wang, T. Schaul, M. Hessel, et al. Dueling network architectures for deep reinforcement learning. arXiv preprint [arXiv: 1511.06581](https://arxiv.org/abs/1511.06581) (2015)
17. Y. Ouyang, Task offloading algorithm of vehicle edge computing environment based on Dueling-DQN, in *Journal of Physics: Conference Series*, Vol. 1873. No. 1. IOP Publishing, (2021)
18. S. Song, Z. Fang, Z. Zhang, C. Chen, H. Sun, Semi-online computational offloading by dueling deep-Q network for user behavior prediction. *IEEE Access* **8**, 118192–118204 (2020). <https://doi.org/10.1109/ACCESS.2020.3004861>
19. F. Jiang, R. Ma, C. Sun, Z. Gu, Dueling deep Q-network learning based computing offloading scheme for F-RAN, in *IEEE 31st Annual International Symposium on Personal, Indoor and Mobile Radio Communications*, London, UK 2020, 1–6 (2020). <https://doi.org/10.1109/PIMRC48278.2020.9217355>
20. F. Jiang, Z. Yuan, C. Sun, J. Wang, Deep Q-learning-based content caching with update strategy for fog radio access networks. *IEEE Access* **7**, 97505–97514 (2019). <https://doi.org/10.1109/ACCESS.2019.2927836>
21. J. Zhang, W. Xia, F. Yan, L. Shen, Joint computation offloading and resource allocation optimization in heterogeneous networks with mobile edge computing. *IEEE Access* **6**, 19324–19337 (2018)
22. Y. Wei, Z. Wang, D. Guo, F.R. Yu, Deep q-learning based computation offloading strategy for mobile edge computing. *Comput. Mater. Continua* **59**(1), 89–104 (2019). <https://doi.org/10.32604/cmc.2019.04836>

23. L. Zhang, B. Cao, Y. Li, M. Peng, G. Feng, A multi-stage stochastic programming-based offloading policy for fog enabled IoT-eHealth. *IEEE J. Sel. Areas Commun.* **39**(2), 411–425 (2021). <https://doi.org/10.1109/JSAC.2020.3020659>
24. A.A. Majeed, P. Kilpatrick, I. Spence, B. Varghese, Modelling fog offloading performance, in *2020 IEEE 4th International Conference on Fog and Edge Computing (ICFEC)*, Melbourne, VIC, Australia, 2020, pp. 29–38, <https://doi.org/10.1109/ICFEC.2020.00011>
25. M. Tang, V.W.S. Wong, Deep reinforcement learning for task offloading in mobile edge computing systems. *IEEE Trans. Mob. Comput.* (2020). <https://doi.org/10.1109/TMC.2020.3036871>
26. Y. Li, F. Qi, Z. Wang, X. Yu, S. Shao, Distributed edge computing offloading algorithm based on deep reinforcement learning. *IEEE Access* **8**, 85204–85215 (2020). <https://doi.org/10.1109/ACCESS.2020.2991773>
27. D. Van Le, C. Tham, A deep reinforcement learning based offloading scheme in ad-hoc mobile clouds, in *IEEE INFOCOM 2018—IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, Honolulu, HI, (2018), pp. 760–765, <https://doi.org/10.1109/INFOCOMW.2018.8406881>
28. C. Huang, P. Chen, Joint demand forecasting and DQN-based control for energy-aware mobile traffic offloading. *IEEE Access* **8**, 66588–66597 (2020). <https://doi.org/10.1109/ACCESS.2020.2985679>
29. Z. Wei, B. Zhao, J. Su, X. Lu, Dynamic edge computation offloading for internet of things with energy harvesting: a learning method. *IEEE Internet Things J* **6**(3), 4436–4447 (2019). <https://doi.org/10.1109/JIOT.2018.2882783>
30. Y. Huang, G. Wei, Y. Wang, V-D D3QN: the variant of double deep Q-learning network with dueling architecture, in *2018 37th Chinese Control Conference (CCC)*, Wuhan, (2018), pp. 9130–9135, <https://doi.org/10.23919/ChiCC.2018.8483478>
31. B.-A. Han, J.-J. Yang, Research on adaptive job shop scheduling problems based on dueling double DQN. *IEEE Access* **8**, 186474–186495 (2020). <https://doi.org/10.1109/ACCESS.2020.3029868>
32. H. Sasaki, T. Horiuchi, S. Kato, A study on vision-based mobile robot learning by deep Q-network, in *2017 56th Annual Conference of the Society of Instrument and Control Engineers of Japan (SICE)*, Kanazawa, (2017), pp. 799–804, <https://doi.org/10.23919/SICE.2017.8105597>
33. H. Ge, Y. Song, C. Wu, J. Ren, G. Tan, Cooperative deep Q-learning with Q-value transfer for multi-intersection signal control. *IEEE Access* **7**, 40797–40809 (2019). <https://doi.org/10.1109/ACCESS.2019.2907618>
34. K. Elgazzar, P. Martin, H.S. Hassanein, Cloud-assisted computation offloading to support mobile services. *IEEE Trans. Cloud Comput.* **4**(3), 279–292 (2016). <https://doi.org/10.1109/TCC.2014.2350471>
35. P. Ajay Rao, B. Navaneesh Kumar, S. Cadabam, T. Praveena, Distributed deep reinforcement learning using tensorflow, in *2017 International Conference on Current Trends in Computer, Electrical, Electronics and Communication (CTCEEC)*, Mysore, (2017), pp. 171–174, <https://doi.org/10.1109/CTCEEC.2017.8455196>

Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Fan Jiang received the B.S. degree and the M.S. degree in communication engineering and communications and information systems from Xidian University, Xi'an, China, in 2004 and 2007, respectively, and the Ph.D. degree in circuit and system from Beijing University of Posts and Telecommunications, Beijing, China in 2010. She is currently a full professor with Xi'an University of Posts and Telecommunications. Her research interests include wireless communication systems, such as device-to-device communications, fog computing, edge caching, and radio resource allocation and management.

Rongxin Ma received the B.E. degree in communication engineering from the Xi'an University of Posts and Telecommunications, Xi'an, China, in 2018, where she is currently pursuing the M.S. degree. Her research interests include reinforcement learning, optimization, and its application in fog computing and computing offloading strategy.

Youjun Gao received the Ph.D. degree in Circuit and System from the Beijing University of Posts and Telecommunications, Beijing, China, in 2008. He is currently with China Mobile System Integration Co., Ltd. His research interests include key technology for future wireless communication, and the applications in smart city and internet of things field.

Zesheng Gu received the B.E. degree in electrical and information engineering from the Lanzhou Polytechnical College, Lanzhou, China in 2016. He is currently pursuing the M.S. degree in electronic communications engineering in the Xi'an University of Posts and Telecommunications, Xi'an, China. His research interests include reinforcement learning, optimization, and Non-orthogonal Multiple Access in the 5G technology.