Computer Communications 160 (2020) 14–24

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

"DRL + FL": An intelligent resource allocation model based on deep reinforcement learning for Mobile Edge Computing

Nanliang Shan *, Xiaolong Cui, Zhiqiang Gao

College of Information Engineering, Engineering University of PAP, Xi'an 710086, China

ARTICLE INFO

Keywords:
Mobile edge computing
Intelligent resource allocation
Deep reinforcement learning
Federated learning

ABSTRACT

With the emergence of a large number of computation-intensive and time-sensitive applications, smart terminal devices with limited resources can only run the model training part of most intelligent applications in the cloud, so a large amount of training data needs to be uploaded to the cloud. This is an important cause of core network communication congestion and poor Quality-of-Experience (QoE) of user. As an important extension and supplement of cloud computing, Mobile Edge Computing (MEC) sinks computing and storage resources from the cloud to the vicinity of User Mobile Devices (UMDs), greatly reducing service latency and alleviating the burden on core networks. However, due to the high cost of edge servers deployment and maintenance, MEC also has the problems of limited network resources and computing resources, and the edge network environment is complex and mutative. Therefore, how to reasonably allocate network resources and computing resources in a changeable MEC environment has become a great aporia. To combat this issue, this paper proposes an intelligent resource allocation model "DRL + FL". Based on this model, an intelligent resource allocation algorithm DDQN-RA based on the emerging DRL algorithm framework DDQN is designed to adaptively allocate network and computing resources. At the same time, the model integrates the FL framework with the mobile edge system to train DRL agents in a distributed way. This model can well solve the problems of uploading large amounts of training data via wireless channels, Non-IID and unbalance of training data when training DRL agents, restrictions on communication conditions, and data privacy. Experimental results show that the proposed "DRL + FL" model is superior to the traditional resource allocation algorithms SDR and LOBO and the intelligent resource allocation algorithm DRLRA in three aspects: minimizing the average energy consumption of the system, minimizing the average service delay, and balancing resource allocation.

1. Introduction

In recent years, with the continuous development of smart terminal devices, people have continuously increased the diversity and quality requirements of mobile services. A large number of computation-intensive and time-sensitive applications have appeared in mobile edge networks, such as object detection, gesture recognition, 3D modeling, interactive games, etc. However, due to the limited resources of network and computing, smart terminal devices can only run the model training part of most intelligent applications in the cloud and force a large amount of training data required by intelligent applications to be uploaded to the cloud, causing congestion in core network communications and reducing the Quality-of-Experience (QoE) of user.

As an important extension and supplement of cloud computing, a new computing paradigm—Mobile Edge Computing (MEC) [1], which sinks computing and storage resources from the cloud to the vicinity of User Mobile Devices (UMDs) to alleviate the burden on the core network. At the same time, offloading computing-intensive and time-sensitive applications on mobile devices to edge servers can reduce the energy consumption of UMDs and the service delay of application requests, and improve the Quality-of-Service (QoS) for mobile users. However, network resources and computing resources are also limited in the MEC system. If a large number of computing tasks cannot be efficiently transmitted to the edge servers, it will also cause congestion of wireless channels and lack of computing resources of the edge servers. Therefore, how to adaptively jointly allocate network resources and computing resources in the MEC system is the key to supporting computing offload and providing high service quality.

Most existing works use multi-objective optimization, semi-positive definite relaxation (SDR) and game theory methods to realize resource allocation. Even though these jobs can achieve better results in specific hypothetical scenarios. Nevertheless, considering the actual edge network scenario in MEC, these optimization methods may be difficult to cope with the following problems: (1) Dynamic network environment;

* Corresponding author.
E-mail address: nanliang@stu.xmu.edu.cn (N. Shan).

https://doi.org/10.1016/j.comcom.2020.05.037
Received 31 December 2019; Received in revised form 22 April 2020; Accepted 25 May 2020
Available online 28 May 2020

dynamic network environment means uncertain user requests for input and changing system states, and these changing system conditions will have a significant impact on network and computing load distribution. (2) Lack of continuity; in a highly time-varying MEC system, most existing resource allocation algorithms can only achieve system optimization at a certain moment. In a real environment, we need to face continuous variables and continuous states. Only through continuous control can long-term benefits be brought to the system.

The use of Artificial Intelligence (AI) techniques to optimize the resource allocation process is a current new trend. Related research on computing offloading and resource allocation, such as [2,3], has proven Reinforcement Learning [4] (especially Deep Reinforcement Learning (DRL) [5]) has unprecedented potential in joint resource management. Therefore, this paper utilizes DRL to adaptively allocate network and computing resources. In addition, Federal Learning (FL) [6] is also introduced as a framework for training of DRL agents in a distributed way while (1) greatly reducing the amount of data uploaded through the wireless uplink channel, (2) increasing cognitive response to edge network and core network environments, (3) adapting well to the heterogeneous UMDs in the actual edge network, and (4) protecting the privacy of personal data [7]. Our contributions are summarized as follows:

(1) We use the DRL architecture (specifically, DDQN [8]) to optimize the network and computing resource allocation of mobile edge systems, and propose a DDQN based Network and Computing Resource Allocation (DDQN-RA) algorithm, which can obtain perceptual information from the environment to improve strategies to adapt to changing circumstances and make decision sequences to realize adaptive resource allocation.

(2) We propose a "DRL + FL" resource allocation model, applying a Federal Learning framework to deploy DRL agents in edge networks. The FL framework trains DRL agents in a distributed way. The advantage of this combination is that it complements the advantages of deep reinforcement learning and federated learning to achieve approximately optimal performance.

(3) We consider minimizing the average system energy consumption and average service delay of all requests made by UMDs and balancing the network load on each data link and the computing load on the MEC servers. Through our proposed "DRL + FL" model, the intelligent joint management of the network and computing resources in the MEC system is realized. And it proves that this scheme has advantages in average system energy consumption, average service delay, and load balancing.

The rest of this paper is organized as follows: We review the related work in Section 2. Then, we depict the system model in Section 3, followed by the problem formulation in Section 4. In Section 5, we optimize the resource allocation by DRL. Furthermore, in Section 6, we integrate FL with DRL in the MEC system. Performance evaluation is shown in Section 7 compared with some classical algorithms. Finally, in Section 8, we conclude this paper.

2. Related work

Mobile edge computing, as a supplement to cloud computing, solves the problems of limited mobile device resources and request–response delays in mobile edge systems through computing offload strategies and mobile application deployment strategies. In order to solve the problems of mobile edge network congestion and edge server load imbalance, it is urgent to use a resource allocation method to allocate network and computing resources in the MEC system to improve the service efficiency of the system, the load balance of the edge server nodes and the standby time of UMDs.

Traditional resource allocation methods face huge challenges in responding to the dynamically changing MEC environment and long-term benefits. For example, literature [9] uses semi-positive definite relaxation (SDR) to jointly optimize communication resource allocation and offloading decisions, and literature [10] uses game theory to jointly optimize equipment delay and energy consumption. However, the above method does not take into account the dynamics of task arrival and system environment, and can only achieve system optimization at a certain moment. To solve these problems, some methods that use machine learning (especially, reinforcement learning) to solve the problem of resource allocation have emerged as the times require [11]. Because reinforcement learning is to maximize long-term returns, it is very suitable for dynamically changing MEC systems, which can adaptively allocate multiple resources. In order to optimize energy consumption and delay, literature [12] proposed a scheme based on reinforcement learning, which minimizes energy and delay consumption by jointly optimizing the allocation of computing resources and network resources. In addition, under the average cost criterion, literature [13] transformed the resource allocation model into a semi-Markov decision process and used linear programming techniques to solve the optimization problem. In literature [14], a resource allocation scheme based on deep reinforcement learning was proposed, which can reduce the average service delay of the system and balance the use of resources.

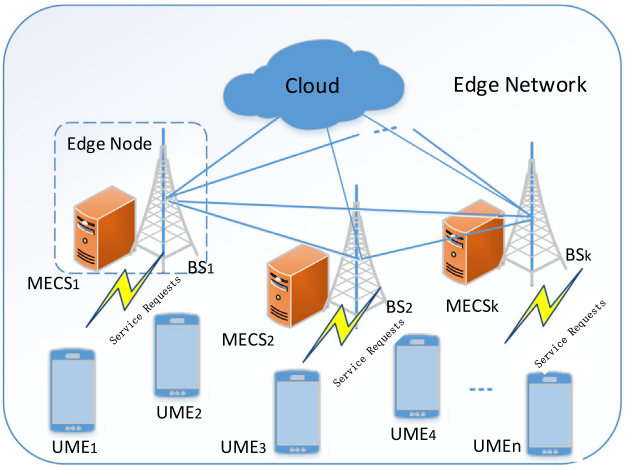

Fig. 1. The MEC-enabled mobile edge system.

Although the above work can achieve ideal results in a dynamic MEC system environment, none of these works has considered: (1) in what form should the training data be gathered (distributed or centralized way)? (2) Where should reinforcement learning agents be placed and trained (at the terminal, edge node or remote cloud infrastructure)? (3) How should the reinforcement learning agents in the update process cooperate? (4) How to solve the problem of overestimation in deep reinforcement learning? And (5) How to protect the privacy of training data? In this paper, we take full advantage of the characteristics of DRL and FL and propose an intelligent resource allocation scheme "DRL + FL", which uses advanced artificial intelligence methods to adaptively allocate network and computing resources under different MEC conditions. It provides a new idea for solving the above problems.

3. System model

3.1. Network model

As shown in Fig. 1, considering a scenario in which a group of UMDs $N = \{1, \ldots, n\}$ is covered by a group of base stations $K = \{1, \ldots, k\}$, UMDs can choose to offload the compute-intensive tasks to the edge node through a wireless channel or execute tasks locally. The compute-intensive task I_n is defined as $I_n(B_n, D_n)$, where B_n is the data amount that the user needs to complete the task [15], and D_n is the number

of CPU clock cycles required to complete the task [16]. In addition, all channels of a base station are defined as the set $M = \{1, \ldots, m\}$, and all virtual machines on a MEC server are defined as $V = \{1, \ldots, v\}$ (assuming that each application request will be allocated to execute on a separate virtual machine). Specifically, the offloading decision $a_n = \{0\} \cup M$ indicates that the computing task can be offloaded to the edge node through the wireless channel a_n or the task can be completed locally ($a_n = 0$). In order to simulate the change of the wireless channel, the channel gain state between the UMDs and the base station is independently selected from the finite state space, and the channel state transition process is modeled as a finite state discrete-time Markov chain [17]. In this wireless scenario, the Shannon theorem [18] can be used to evaluate the achievable data transmission rate. The channel capacity calculated by Shannon theorem is the maximum available data transmission rate. Usually, the actual data transmission rate is less than the channel capacity. Using Shannon theorem to evaluate the achievable data transmission rate, the minimum delay of data transmission can be obtained. This minimum delay value can be used as a threshold for offloading decisions.

3.1.1. Network transmission

When $a_n \in M$, assuming $a_n = m$, UMD offloads compute-intensive tasks to edge nodes through wireless channel m, and the transmission delay and energy consumption of the application program on user n is offloaded to virtual machine v on server k through wireless channel m are defined as:

$$t_{n,m}^{k,v} = \frac{B_n}{c} \tag{1}$$

and

$$e_{n,m}^{k,v} = \frac{q_n B_n}{c} + T_n \tag{2}$$

Among them, c is the data transmission rate in the channel, q_n is the transmission power between user n and the base station, and T_n is the trailing energy generated in the wireless transmission.

3.1.2. Variance of network resource allocation

In order to alleviate edge network congestion and ensure the successful transmission of requests of compute-intensive applications on mobile devices, network resource allocation on the data link should be balanced. Let l_m^{net} denote the load of channel m in the edge network. Therefore, the variance in network resource allocation in the entire edge network can be expressed by the disparity in network load and defined as:

$$var(l^{\text{net}}) = \frac{\sum_{m \in M} (l_m^{\text{net}} - \frac{\sum_{m \in M} l_m^{\text{net}}}{|M|})^2}{|M|} \tag{3}$$

3.2. Computing model

The computing model includes two parts: local computing and MEC server-side computing.

3.2.1. Local computing

When edge user decides to perform local computing, the edge user n performs the computing task I_n locally. Assume f_n^l is the computation capability of the edge user (the clock frequency unit of the edge user CPU runs is HZ). Where γ_{n}^l is the coefficient denoting the consumed energy per CPU cycle of the UMD, which can be obtained by the measurement method in [19]. So we can get the execution time and energy consumption of the local computation task I_{n} as:

$$t_{n,m}^l = \frac{D_n}{f_n^l} e_{n,m}^l = \gamma_n^l D_n \tag{4}$$

3.2.2. MEC server-side computing

On the MEC server-side, the computation capability of the edge server is the clock frequency f_n^c. Where γ_{n}^c is the coefficient denoting the consumed energy per CPU cycle of the edge server, which can be obtained by the measurement method in [19]. Then the execution time and energy consumption of computing tasks I_n performs on the edge server node can be given as:

$$T_{n,m}^{k,v} = \frac{D_n}{f_n^c}, E_{n,m}^{k,v} = \gamma_n^c D_n \tag{5}$$

3.2.3. Variance of computing resource allocation

In order to avoid overloading the computing load on a single MEC server and causing request timeouts, the computing resource allocation between each edge node and the remote cloud should be balanced to ensure that application requests on UMDs can get service responses with minimal delay. The load of MEC server k denoted as l_k^{cp}. Therefore, the variance in computing resource allocation in the entire edge network can be expressed by the disparity in the computing load and defined as:

$$var(l^{cp}) = \frac{\sum_{k \in K} (l_k^{cp} - \frac{\sum_{k \in K} l_k^{cp}}{|K|})^2}{|K|} \tag{6}$$

3.3. Service model

Due to the high complexity of the MEC system, when a large number of users are involved, it is very difficult to obtain the information needed globally to deal with the offloading of computing tasks and resource allocation. Therefore, this paper uses Double Deep Q Network (DDQN) [20], the most representative and effective algorithm framework in DRL, to jointly control the resource allocation behavior of MEC system. The service model operation mechanism is summarized as follows: UMDs determine the joint allocation of wireless channels and computing resources according to a stationary control strategy $\boldsymbol{\Phi} = (\boldsymbol{\Phi}_{\text{net}}(\xi), \boldsymbol{\Phi}_{\text{cp}}(\xi))$, which definition is to map the state to the probability distribution of the actions, represents the actions of the agent and directs the agent how to choose action. At the same time, the system state ξ needs to be continuously observed, including the task queue status, the location generated by the request, the location of the available server, the information of the task to be offloaded, the channel occupancy situation, and the remaining computing resource of the server. In addition, we also define an immediate utility function $u(\xi, (\text{net}, \text{cp}))$ to evaluate the QoE of UMDs, which is inversely proportional to the number of tasks lost and failed, the energy consumption of task execution, the task execution delay (including wireless transmission delay and computation delay), task queue delay and the variance of network and computing resource allocation. By using the Double Deep Q-Learning algorithm, the control strategy $\boldsymbol{\Phi} = (\boldsymbol{\Phi}_{\text{net}}(\xi), \boldsymbol{\Phi}_{\text{cp}}(\xi))$ can be trained and the long-term optimization of UMDs service performance can be achieved.

3.3.1. Reinforcement learning

Reinforcement learning (RL) is a branch of machine learning, which focuses on acquiring knowledge in the environment, adapting to the environment by improving behavior strategies and making sequence decisions. RL system assumes that there is an agent in the environment to implement the actions. By exploring the environment and receiving feedback, RL system can form an adaptive model without a large amount of labeled data. Intuitively speaking, RL is a process in which agents continuously interact with the environment to make sequential decisions and enhance their decision-making capabilities. In the scenario we considered, through continuous interaction with the MEC environment, agents can take action and receive corresponding rewards. In addition, its goal is to maximize the cumulative reward. As shown in Fig. 2, for each episode, first, at each step t, the agent acquires the observation of the MEC environment, that is, the state s_t.

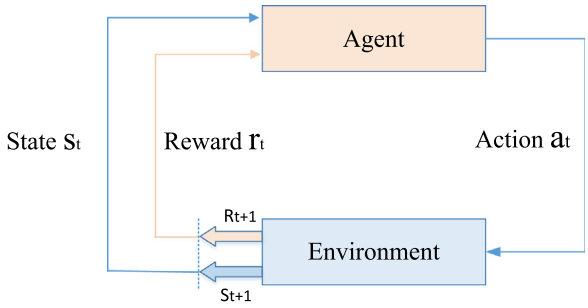

Fig. 2. The illustration of reinforcement learning.

Then, the agent makes an action at based on a certain policy, obtains the corresponding reward r_t, and enters a new state s_{t+1}. Subsequently, the policy is updated based on the reward given by the environment. Q-learning is one of the classic reinforcement learning algorithms and a model-free learning method, which uses the action value $Q(s_t, a_t)$ stored in the Q-table to choose action.

3.3.2. Deep reinforcement learning

Deep reinforcement learning is the integration of deep learning and reinforcement learning, which can perfectly combine the perception ability of deep learning with the decision-making ability of reinforcement learning. Deep Q Network (DQN) is the most representative framework of DRL. Traditional Q-learning uses Q-table to store action values, and in the scenario we consider, the state of the MEC environment is complex and diverse. Therefore, it is impractical to use a table to store all the action values and also time-consuming to search for the corresponding state in a large table frequently. Deep Q Network directly uses the neural network with parameter ω to approximate the Q function and generate action values. However, DQN uses the same Q value to select and evaluate an action, which leads to an overestimation of the Q value [20]. The author proposed an upgraded version of DQN in [20], Double Deep Q Network (DDQN), which uses two Q networks to fit Q value functions and randomly assigns experience to update one of the two Q value functions. Therefore, two different weights can be generated, one to determine the greedy policy and the other to determine its value. This method can realize the decoupling of selection and evaluation, which solves the problem of overestimation.

The specific framework is shown in Fig. 3, DDQN uses a neural network with parameters ω to approximate the Q-value function directly and generate action values $Q(s, a; \omega)$. The input of the neural network is the state s, and the output is the action value $Q(s, a; \omega)$. In DDQN, it first selects the action corresponding to $\max_{a_t} Q(s_t, a_t; \omega)$ according to the Q-value output by MainNet, and then evaluate the Q-value of this action in TargetNet. We use the ε-greedy strategy [21] to select the action. The probability of ε-greedy strategy randomly picking an action is $\varepsilon \in (0, 1)$, and the probability of selecting action $a = \arg \hat{Q}(s_t, \arg\max_{a_t} Q(s_t, a_t; \omega); \omega^-)$ is $1 - \varepsilon$. In this way, the DRL agent can not only maximize the rewards of known information but also explore the information of the unknown environment. Therefore, state–action pairs that are not in the sample can also be learned.

Neural network training requires a loss function optimization process, which minimizes the deviation between the label and the output result. The parameters of the neural network will be updated by back-propagation and gradient descent [22]. The neural network in DDQN is no exception. The goal of DDQN is to make the Q-value close to the target Q-value and use the Double Q-learning algorithm to provide so-called labels. This paper uses the Mean Square Error (MSE) [23] as the loss function of the neural network in DDQN, which is defined as:

$$L(\omega) = E[(\text{Target}Q - Q(s_t, a_t; \omega))^2] \tag{7}$$

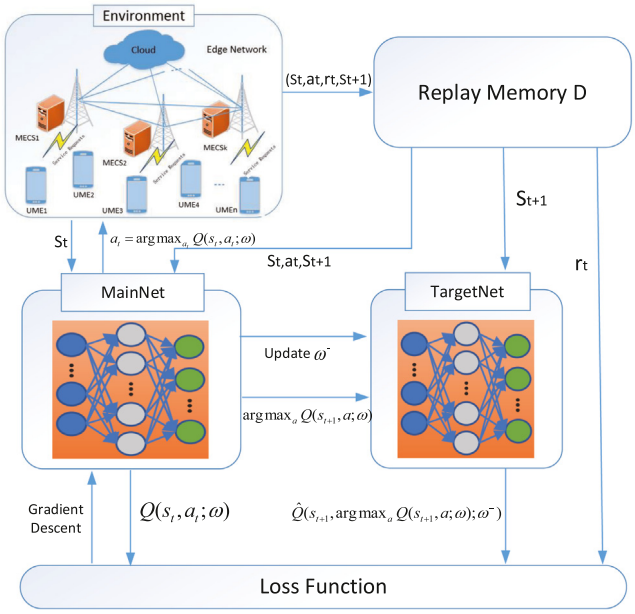

Fig. 3. A service model framework of double deep Q network.

among

$$\text{Target}Q = r + \gamma \hat{Q}(s_{t+1}, \arg\max_{a_{t+1}} Q(s_{t+1}, a_{t+1}; \omega); \omega^-) \tag{8}$$

In Eq. (8), Target Q is calculated by TargetNet with parameter ω^- and $Q(s_t, a_t; \omega)$ is the evaluation Q-value output by MainNet with parameter ω. As shown in Fig. 3, TargetNet is used to provide immovable labels to improve training stability and convergence. The initial parameters of TargetNet are the same as MainNet. However, ω is updated every step, and ω^- is updated every stationary C steps. Therefore, the parameter update speed of TargetNet is slower than MainNet, which guarantees the stability of the system.

4. Problem formulation

From the description in Section 3, the system energy consumption includes the transmission energy consumption in the edge network and the computing energy consumption of the server-side (the transmission energy consumption of the local computing can be considered as 0), and the service delay includes the data transmission delay in the edge network and corresponding server data processing delay. This paper focuses on designing an intelligent allocation algorithm for network and computing resources based on MEC. Our goal is to minimize the average system energy consumption and average service latency of all requests made by mobile devices and balance the computing load on each MEC server and the network load on the data link. Then, the QoS and adaptability of the MEC architecture will be improved.

Average energy consumption minimization: As mentioned above, the system energy consumption includes transmission energy consumption in the edge network and server-side computing energy consumption. In order to save energy costs and provide longer standby services at the edge, our goal is to use artificial intelligence techniques to minimize the average energy consumption of the entire MEC system. As described above, $e_{n,m}^{k,v}$, $e_{n,m}^{l}$ and $E_{n,m}^{k,v}$ are defined as the energy consumption for data transmission in the edge network, the energy consumption for data processing of local computing, and the energy consumption for computing data processing on the MEC server respectively. The energy consumption performed by the application program on the user n to the virtual machine v on the server k through the wireless channel m can be expressed as $e_{n,m}^{k,v} + E_{n,m}^{k,v}$. Therefore, the

problem of minimizing average energy consumption of users can be defined as:

$$\min \frac{\sum_{n\in N}\sum_{m\in M}\sum_{k\in K}\sum_{v\in V}\left[\tau\cdot e^{l}_{n,m}+(1-\tau)\cdot(e^{k,v}_{n,m}+E^{k,v}_{n,m})\right]}{N_{\text{total}}}, \forall\tau\in\{0,1\} \quad (9)$$

Where τ indicates whether the application request is executed locally or offloaded to the MEC server-side, and N_{total} denotes the total number of application requests generated by the UMDs in the edge network.

Average service delay minimization: As mentioned above, the service delay includes the data transmission delay in the edge network and the corresponding server-side data processing delay. In order to improve QoS and better support the advanced functions of mobile devices, our goal is to minimize the average service delay of all requests generated by mobile devices distributed in different regions in an adaptive manner. As described above, $t^{k,v}_{n,m}$, $t^{l}_{n,m}$ and $T^{k,v}_{n,m}$ are defined as the data transmission delay in the edge network, the data processing delay of local computing, and the data processing delay of MEC server-side computing respectively. The service delay performed by the application program on the user n to the virtual machine v on the server k through the wireless channel m can be expressed as $t^{k,v}_{n,m}+T^{k,v}_{n,m}$. Therefore, the problem of minimizing the average service delay of users can be defined as:

$$\min \frac{\sum_{n\in N}\sum_{m\in M}\sum_{k\in K}\sum_{v\in V}\left[\tau\cdot t^{l}_{n,m}+(1-\tau)\cdot(t^{k,v}_{n,m}+T^{k,v}_{n,m})\right]}{N_{\text{total}}}, \forall\tau\in\{0,1\} \quad (10)$$

Resource allocation balancing: Application requests generated by UMDs will be offloaded to the corresponding MEC servers for service according to the offload decision. Application request generation rates in different regions are different. As the number of users increases, if application requests are offloaded to the same region, it will put tremendous pressure on network links and MEC servers in this region. The excessive load of the selected network link and the MEC server may lead to a large transmission delay and processing delay so that the overall service delay and system energy consumption will be greatly increased. At the same time, network link congestion and increased service delay will lead to an increased probability of application requests being lost and application requests failing. Therefore, balancing the network load across data links and the computing load of MEC servers to achieve a balanced allocation of resources is of great significance for reducing system energy consumption, average service delay, and alleviating the computing pressure of MEC servers. With this, the adaptability of the MEC architecture will also be improved. The resource allocation balancing problem is to minimize the variance of the network load on each link and minimize the variance of the computation load on each MEC server (i.e., $\min Z = \text{var}(l^{\text{net}}) + \text{var}(l^{cp})$).

5. Optimizing the resource allocation by DRL

Considering that the dynamic changes of the environment will cause uncertain inputs and changes in system conditions, intelligent resource allocation algorithms should consider the state of the MEC environment to make favorable decisions. Deep reinforcement learning systems can form adaptive models without a large amount of labeled data by exploring the environment and receiving feedback. It is very suitable for the changing MEC environment we are considering. Therefore, in this section we propose a DDQN based Network and Computing Resource Allocation (DDQN-RA) algorithm, which minimizes system average energy consumption, minimizes average service delay and balances resource allocation in an adaptive manner.

5.1. Training

Several pioneer works have solved some problems through resource allocation algorithms based on traditional Q-Learning or DQN architecture. Traditional Q-Learning uses Q tables to store Q-values, which is not feasible for actual MEC systems due to the actual state–action space is continuous and huge [24]. It is impractical to store all Q-values in a table and also time-consuming to search for information in a large table frequently. The DQN framework uses the Deep Q-learning algorithm, which still cannot overcome the inherent shortcomings of Q-learning itself—overestimation [25], that is, the estimated Q-value is larger than the real value and the estimation error increases with the number of iterations. The resource allocation algorithm DDQN-RA proposed in this paper is based on Double Deep Q-Learning. It uses a neural network to approximate the Q-value function directly to choose the actions and the evaluate the actions with another neural network, which can solve the overestimate problem effectively. The three elements of deep reinforcement learning used in our algorithm, i.e., state, action, and reward, need to be defined first:

State: The state of our proposed DDQN-RA algorithm includes the location generated by the requests, the location of the available servers, the information of the tasks to be offloaded, the channel occupancy situation, and the remaining computing resource of the servers. The state vector can be expressed as $s = \{n, k', I, \mu, \xi\}$, where n is defined as the location of the user who generated the application requests, k' is defined as the set of available servers location, and I is defined as the task information to be offloaded (including the size of offloaded data and the number of CPU clock cycles required to complete the task) , μ is defined as the occupancy of the channels corresponding to the available server, and ξ is defined as the remaining computing resource availability of the servers.

Action: In the DRL system, actions correspond to the states and the current action will affect the next state. The action vector can be expressed as $a = \left\{\alpha^{k,v}_{n,m}\forall n\in N, \forall m\in M, \forall k\in K, \forall v\in V\right\}$, and $\alpha^{k,v}_{n,m}$ is defined as an action performed when the application request of user n is offloaded to virtual machine v on MEC server k through channel m.

Reward: In each step t, after taking a possible action a_t, the DRL agent can get a reward from the environment. The reward needs to reflect the purpose of our proposed intelligent resource allocation algorithm DDQN-RA, which is to minimize the system average energy consumption, minimize the average service delay and balance the allocation of network and computing resources. We define the reward received by the DRL agent from the environment as $r = \lambda_1\cdot E+\lambda_2\cdot T+\lambda_3\cdot b^n+\lambda_4\cdot b^c$, where E is defined as the system average energy consumption of the application request, T denotes the service delay of the application request, and b^n represents the balance degree of network resource allocation in the entire MEC network, b^c represents the balance degree of computing resource allocation on all MEC servers. b^n and b^c can be obtained according to the variance of the transmission load of each network channel var(l^{net}) and the variance of the computing load of each MEC server var(l^{cp}). In addition, $\lambda_i, i \in 1,2,3,4$ represents the weight of the corresponding object in the reward expression.

5.2. Running

As shown in Fig. 3 and Algorithm 1, we show the detailed process of intelligent resource allocation with DRL. We first initialize the experience replay memory D with certain capacity N as well as the MainNet and TargetNet with random parameters ω and ω^{-}. The episode refers to the resource allocation process of offloading an application request to a specific virtual machine on the target MEC server. For each episode k, we first initialize state s, and then for each step t, state s_t will be used as the input to MainNet. In the DDQN framework, we first find the action corresponding to the MainNet's output max Q-value, and then use TargetNet to estimate the target Q-value of this action. This can avoid overestimating the Q-value. The action a_t will be selected through the target $\hat{Q}$ value based on the ϵ-greedy strategy. After the action a_t is selected, the corresponding reward r will be obtained and the system will enter the next state s_{t+1}. We store the corresponding data items (s_t, a_t, r_t, s_{t+1}) in experience replay memory D to update MainNet. In the update process of MainNet, we first randomly

Algorithm 1 DDQN based Resource Allocation (DDQN-RA)Algorithm

Require: Discount rate γ, exploration rate ε, replay memory capacity N
Initialize replay memory D to capacity P
Initialize MainNet with parameters ω
Initialize TargetNet with parameters $\omega^- = \omega$
for each episode k **do**
  Initialize state s_1
  for each step t **do**
    Generate random number $\eta \in [0, 1]$
    if $\eta < \varepsilon$ **then**
      Randomly select an action a_t
    else
      Select $a_t = \arg \hat{Q}(s_t, \arg\max_{a_t} Q(s_t, a_t; \omega); \omega^-)$, where max Q is estimated by MainNet, $\hat{Q}$ is evaluated by TargetNet
    end if
    Execute action a_t in emulator
    Observe reward r_t and new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in D
    Sample random minibatch of transitions (s_t, a_t, r_t, s_{t+1}) from D
    Set $y_t = \begin{cases} r_t, & \text{if episode terminates at step } t+1 \\ r_t + \gamma \hat{Q}(s_{t+1}, \arg\max_{a_{t+1}} Q(s_{t+1}, a_{t+1}; \omega); \omega^-), & \text{otherwise} \end{cases}$
    Execute gradient descent using MSE function $(y_t - Q(s_t, a_t; \omega))^2$ with respect to the parameters ω and ω^-
    Each C steps reset $\omega^- = \omega$
  end for
end for

sample a small batch of data items from the replay memory D to train the neural network. This is helpful to break the connection between samples and ensure the independent and identical distribution (IID) of training samples. Therefore, the evaluation Q-value $Q(s_t, a_t; \omega)$ and target Q-value $\gamma \hat{Q}(s_{t+1}, \arg\max_{a_{t+1}} Q(s_{t+1}, a_{t+1}; \omega); \omega^-)$ will be calculated through the MainNet and TargetNet respectively. Subsequently, the parameter ω of MainNet is updated according to the MSE loss function as described in Eq. (7). The parameter ω^- of TargetNet only needs to be updated once every C steps. C is a certain factor controlling the update rate of the target network, and the value of C is only determined by the network structure. Finally, in episode k, the trained MainNet is obtained by continuously repeating step t, where t represents the tth resource allocation method and k represents the kth resource allocation procedure.

It should be noticed that our goals are to minimize system average energy consumption, minimize average service delay, and balance the allocation of network and computing resources. These goals will be reflected in the reward. When the DRL agent takes some good actions in the environment, such as making the system's energy consumption overhead smaller or the average service delay shorter, the DRL agent will get a positive feedback and vice versa. More importantly, when the application request is offloaded to the virtual machine of the corresponding MEC server, the resource allocation balance must also be considered in the calculation of the reward.

6. Integrating FL with DRL in the MEC system

The DDQN-RA algorithm we proposed above needs to be run on the DRL agent. However, the deployment of the DRL agent will be a key issue. Generally, we divide the deployment of DRL agents into three modes, i.e., **Centralized DRL Deployment**, (1)Deploy the DRL agent at the cloud center. And **two modes of Distributed DRL Deployment**, (2) Deploy the DRL agents at the cloud center and edge nodes, and (3) Deploy the DRL agents at the cloud center, edge nodes, and all UMDs. If the first deployment mode is used to train the DRL agent, due to the wireless communication characteristics of the MEC system, there are three main shortcomings: (1) Considering that a large amount of training data is mainly generated by UMDs, it increases the burden on the uplink wireless channel; (2) If the training data uploaded to the edge node or cloud is privacy-sensitive data, it may cause potential privacy leakage risks; (3) If the training data is transformed for privacy consideration, the server-side proxy data is less relevant than on-device data. If the second deployment mode is used to train the DRL agent, in addition to the three shortcomings of the first mode, the problem of non-independent and identical distribution (Non-IDD) and unbalanced distribution [26] of user data will also arise. If the third deployment mode is used to train the DRL agent, although it solves the problem of transmitting a large amount of training data, it also has two shortcomings: (1) the problem of Non-IDD and unbalanced distribution of user data; (2) limited communication conditions cannot guarantee that all UMDs are online at the same time.

DRL techniques require a large amount of data information for finding the optimal solution when dealing with optimization problems of large-scale MEC systems. However, the wireless communication capabilities of MEC systems limit the possibility of efficiently converging all data to the server. Therefore, considering using a distributed deployment of the DRL agent on each UMD, edge node and remote cloud is natural. In this way, a large amount of data generated by the client can be directly used to train the DRL agent without transmitting through the wireless network, which will not generate a lot of transmission delay and energy consumption. However, this distributed DRL agent deployment model will also introduce several issues mentioned above. If these problems can be solved, we will be able to obtain approximately optimal performance. Therefore, Federal Learning (FL) is introduced to train the DRL agents in the MEC system. We call this scheme the "DRL + FL" model and compare the advantages and disadvantages of this method with centralized DRL and conventional distributed DRL, described by Fig. 4. Compared with centralized DRL and conventional distributed DRL, "DRL + FL" has the following advantages:

- Can avoid uploading large amounts of training data: Both centralized DRL and conventional distributed DRL need to upload a large amount of training data from the UMDs to the server, which will greatly increase the burden on the uplink wireless channel.

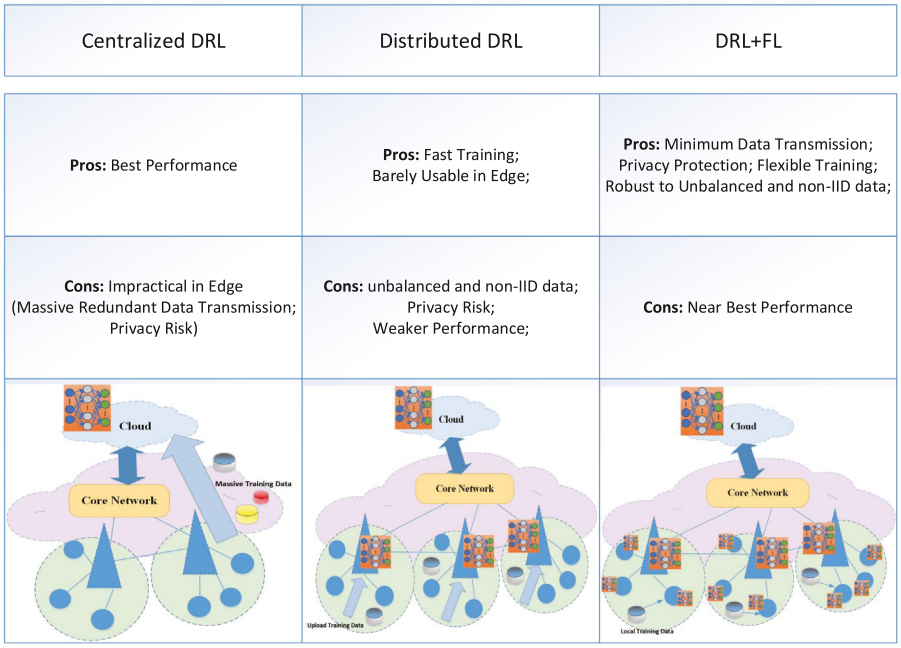

Fig. 4. Taxonomy of deploying DRL agents in mobile edge system.

In the "DRL + FL" model, the UMDs train the DRL agent locally, and only need to upload updated model parameters to the server.

- Can solve the Non-IDD and Unbalanced problem of training data: The training data on the UMDs (stored in the DRL's transition memory) is obtained based on the wireless environment it experiences. And some UMDs may experience more mobile network states. Hence, training data on one UMD alone will not represent the characteristics of all UMDs' training data. In FL, this challenge could be satisfied by merging the models' updates with the FedAvg algorithm in.
- Can reduce the influence of communication conditions: UMDs are often unpredictably off-line or assigned to poor communication resources, which will lead to some model training data transmission is not timely and affect the training effect. However, FL only requires a part of customers to upload their updates in one round of training, which solves the unpredictably off-line situation of customers.
- Can solve the problem of data privacy and security: The information that needs to be uploaded to the FL to improve the DRL agent is the necessary minimum update. Further, techniques of security aggregation and differential privacy can be applied naturally, thereby avoiding uploading sensitive privacy raw data to the server for updates. Nonetheless, privacy security is not the focus of our work, and more information on these issues can be found in Ref. [27,28].

The combination of the DRL method and the FL framework makes the computing offload and resource allocation of the MEC system more intelligent and combines the UMDs with edge nodes and remote clouds to form a powerful intelligent entity. This entity obtains powerful environmental cognitive ability through a large number of UMDs. Each edge node of the entity can support AI tasks on the system level dynamically, which ensures global optimization and balance of the entire MEC system.

6.1. DRL + FL to computation offloading

In the computation offloading scenario, each UMD decides whether to offload a computing task to an edge node, which edge node to offload, and which wireless channel to transmit the offloaded task according to the decision of the DRL agent. Although UMDs such as mobile phones, industrial IoT devices, and smart vehicles are able to execute some AI tasks, computing power and energy consumption still limit their AI computing capabilities (e.g., training large-scale DRL agents). Therefore, we propose to use the "DRL + FL" model to treat all edge service nodes as the central server to coordinate the large number of UMDs they cover. With this solution, UMDs with weak computing ability can accommodate a complex DRL agent and expand computing ability at the edge through computation offloading at the same time.

6.2. DRL + FL to resource allocating

While computation offloading, edge nodes decide whether to allocate network and computing resources to the UMD, which wireless channel is assigned to offload tasks and which virtual machine on which edge server is assigned to execute tasks according to the decision of the DRL agent. The problem of coordination of computation offloading of UMDs and resource allocation of edge nodes has always been a key link affecting user's computing efficiency. Non-intelligent resource allocation methods are always to asynchronous delay. Therefore, we propose the "DRL + FL" scheme to treat all edge service nodes as the central server to coordinate a large number of UMDs they cover. With this solution, the network and computing resource allocation of the edge nodes can be coordinated with the computation offloading of the UMDs and greatly improving the user's computing efficiency.

The methods for dealing with the above two cases are similar. FL iteratively selects a random set of UMDs to (1) download the parameters of DRL agents from edge nodes; (2) uses its own data to perform the training process on the model; (3) upload only updated model parameters of DRL agents to edge nodes for model aggregation. In this way, a shared DRL agent can be trained by FL without having to aggregate training data.

In summary, FL allows resource-constrained edge computing devices (including UMDs and edge nodes) to train shared models while maintaining the locality of the training data. The core advantage of FL is to update the model separately on a large number of devices, without having to aggregate training data.

6.3. Feasibility discussion

To a certain extent, the "DRL + FL" framework we have proposed is a future-oriented concept. We envisage that in the near future, most user mobile devices, especially smartphones, not only have the ability to infer, but also the ability to train lightweight deep learning models. In Sections 6.2 and 6.3, we introduced the use of the FL method to help resource-constrained edge devices train shared models. However, considering the actual deployment and limited communication, FL feasibility should be discussed.

(1) Deploy challenges: Model training usually takes a long time according to its required accuracy requirements. Therefore, the biggest challenge of using FL distributed training deep reinforcement learning model is the deployment of DRL agent. Obviously, while randomly setting the weight of the neural network, the DRL agent should not be directly deployed. Otherwise, the MEC system will be paralyzed because the DRL agent cannot be fully trained on actual mobile devices and can only make random decisions during initial exploration.

However, if the DRL agent is not trained from scratch, this may be resolved. We are now working hard to use transfer learning to facilitate the training process of the MEC system. The basic idea is to simulate the wireless environment and requests of UMDs. Just like evaluating and adjusting antenna settings on a simulation test bed, the simulation environment is used to train an offline deep reinforcement learning agent. Then, the established DRL agent can be distributed to UMDs. This solves the problem of longer model training time on mobile devices.

(2) Limited communication challenges: UMDs are often unpredictable offline or allocated to poor communication resources. Therefore, it is difficult to combine UMDs with edge nodes and remote clouds in the actual environment. In this case, it is difficult to achieve training similar to the distributed shared model, and it will also bring a large transmission delay.

However, FL model integration does not require all users to upload updates at the same time. In a round of integration, only some customers are required to upload their updates, which solves the often unpredictable offline situation of customers. Therefore, in the actual edge network environment, it is also feasible to use FL to combine UMDs with edge nodes and remote clouds.

7. Performance evaluation

7.1. Experiment settings

In order to study the performance of "DRL + FL", we did some simulations on network and computing resource allocation experiments. In all simulations, the time horizon is discretized into time epochs.

On investigating the capabilities of DRL coupled with FL over the MEC system for resource allocation, we set the average request data size within 5MB [29] and the request requirement of CPU cycles between 50 Megacycles and 1 Gigacycles based on the characteristics of several applications in the laboratory test described in [30]. We suppose the bandwidth of an edge node is $\omega = 5$ MHz and the computing ability of MEC is set between 1 GHz and 5 GHz. When all the computing resources of the MEC server are occupied or the remaining computing resources are less than the amount of computation requested by the UMD, the request will be rejected or queued. The request queue size is set to 10, and the request timeout threshold is set to 3s. Within the value of the request queue and not exceeding the timeout threshold, the request is queued, and if the timeout threshold is exceeded, the request is rejected. The average data transmission rate is distributed between 100 Mbps and 450 Mbps according to the work by *Rimal* et al. [31]. The channel gain states between UMDs and edge nodes come from a common finite set, which quantifies the quality of wireless channels into 6 levels. In the whole simulation process, the number of tasks generated by each UMD follows the Bernoulli distribution with average arrival rate λ_T per time epoch.

As for the DRL settings in UMDs, edge nodes, and cloud, we choose the vanilla version of the Double DQN algorithm with parameter settings. We also select tanh as the activation function and Adam optimizer. We use two complete fully connected layers of neural networks as the target (TargetNet) and evaluation (MainNet) Q networks, each layer containing 200 neurons. The other parameter values in Double DQN are set as follows: learning rate $\zeta = 0.005$, discount factor $\gamma = 0.9$, exploration probability $o = 0.001$, replay memory capacity $M = 3000$, minibatch size $B = 32$ and the period of replacing the target Q network is $C = 200$. In addition, in order to establish a performance baseline for the "DRL + FL" model, we take the intelligent resource allocation algorithm DRLRA proposed in [14], and some other resource allocation algorithms, such as the known SDR [32] algorithm and LOBO [33] algorithm for comparison. DRLRA, the resource allocation algorithm based on deep reinforcement learning, differs from the method in this paper in that it does not combine FL, and uses distributed methods to train DRL agents. SDR, a semidefinite relaxation-based optimization approach for task offloading, SDR abstracts the resource allocation problem in the process of computing offloading into a combined optimization problem, and the best solution for resource allocation can be obtained by solving the combined optimization problem. LOBO, a Lyapunov optimization-based online algorithm, obtains the optimal computing load and optimal network load of the edge server through the predefined Lyapunov optimization function. At the same time, in order to make the reference experiment more convincing, we also adopt a centralized DRL [34], which is used for comparison and is assumed to be able to receive all data for reinforcement learning.

7.2. Evaluation results

In order to clarify the performance of our edge intelligence framework "DRL + FL" when performing resource allocation in the MEC system, experiments on network and computing resource allocation were performed under different settings.

Fig. 5 depicts the performance of DDQN with FL in the allocation of network resources and computing resources under the MEC system. Three edge nodes (*k1, k2*, and *k3*) and three UMDs (*n1, n2*, and *n3*) are selected from 5 edge nodes and 50 UMDs to demonstrate the performance of the deployed DRL agent. From the perspective of network resource allocation, the packet loss rates and packet delays of *n1, n2*, and *n3* decrease rapidly with the training process of the DRL agents, and eventually remain stable within a certain range. In the simulation of computing resource allocation, *k1, k2*, and *k3* were selected as servers to execute offloading tasks. With the training process of the DRL agents, the service rejection rate and service delay quickly decreased and eventually remained stable. This means that using the edge intelligent resource allocation model "DRL + FL" in the MEC system can effectively improve the efficiency of resource allocation.

In addition, we give details of the performance comparison of the "DRL + FL" scheme with several other methods as follows.

In order to highlight the impact of the deployment location of DRL agents on resource allocation, we made some assumptions about the system conditions of centralized DDQN. In studying the detailed performance during the training process, we assume that the wireless communication capabilities of the centralized DDQN are not impeded, that is, the process of transmitting a large amount of data for centralized training to the cloud through the core network will not be affected by network traffic.

In Fig. 6, we analyze the impact of the number of service requests and the processing capacity of the MEC servers on the average energy consumption of the MEC system, including data transmission energy consumption, local data processing energy consumption, and MEC server-side data processing energy consuming in the edge network. Here we do not consider the energy consumption of data processing

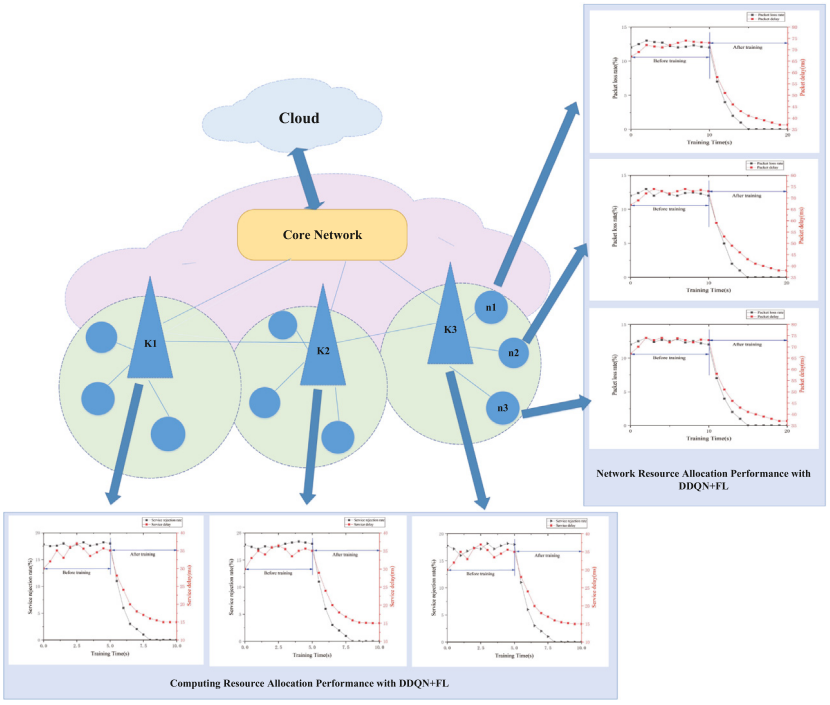

Fig. 5. Performance of "DRL + FL" in network and computing resource allocation.

in the cloud, mainly for two reasons. (1) The energy consumption in the cloud is not part of the edge system. (2) The number of cloud servers is huge and multiple tasks are processed in parallel, so we cannot measure the energy consumption of our tasks. It can be easily seen that with the increase in the number of user service requests and the processing capacity of the MEC server, in terms of optimizing the average energy consumption of the entire MEC system, the intelligent resource allocation method is obviously better than the traditional optimization method, and the performance of "DDQN + FL" is better than all other methods.

In Fig. 7, we analyze the impact of the number of service requests and the processing capacity of the MEC server on the average service delay, including the data transmission delay, the local data processing delay, and the MEC server-side data processing delay in the edge network. It can be seen that the "DDQN + FL" model has a great advantage in terms of average service delay compared with several other resource allocation algorithms. As the number of service requests increases, this advantage becomes more apparent. As the processing capacity of the MEC server increases, the DDQN with FL has a clear advantage in a certain range. But as the processing capacity of the MEC server is sufficient, the advantage is reduced. Therefore, our algorithm will be very practical in a real situation with moderate MEC servers' processing capabilities.

We conduct experiments to verify the network load and computing load balancing issues in the resource allocation process. It can be seen from Fig. 8 that the performance of several intelligent resource allocation algorithms before training is not ideal, but the variance of the network and computing load decreases rapidly after training. After a short training time, our intelligent resource allocation model "DDQN + FL" is always able to obtain stable network and computing load variance, and its performance is slightly better than centralized DDQN. This means that in the process of allocating network and computing resources, once the FL shared model training is completed, "DDQN + FL" will take full account of the load situation in the current network environment. Furthermore, an adaptive allocation decision can be made, which can effectively achieve a balanced allocation of network and computing resources to prevent network congestion and service delay caused by the burst requests. Therefore, if the user wants to use the "DDQN + FL" model to obtain better performance, it must wait for the model to merge to use the training results of other agents.

From the above three comparative experiments, it can be seen that the method of "DDQN + FL" and centralized DDQN can always obtain approximately optimal performance. However, the high performance of the centralized DDQN is obtained under the assumption that we are in an ideal wireless network environment and without considering the energy consumption of the cloud center. In actual situations, as the number of UMDs increases, large amounts of training data transmitted to the cloud through the core network will cause great delay and instability, and at the same time, will also generate a lot of energy consumption. In addition, the energy consumption of the cloud center is usually huge, and we should also consider it from the perspective of resource consumption. Therefore, it is more practical to implement DDQN with FL in the MEC system, at least considering the current core network situation.

As shown in Fig. 9, we also compared the amount of the wireless transmission data of the three intelligent resource allocation models of "DDQN + FL", centralized DDQN, and DRLRA to complete the training process in network and computing resource allocation scenarios. Training DDQN with FL framework, each UMD only needs to upload updated data of its model. Without FL framework, such as centralized DDQN or DRLRA, UMDs must upload all training data to the cloud or exchange all training data at edge nodes via wireless channels. This will cause more communication resources to be consumed, and it will also generate large transmission energy consumption and delay.

8. Conclusions

In this paper, we study the resource allocation problem in a variable MEC environment, including network resource allocation and computing resource allocation. We considered this issue from the aspects of minimizing the average energy consumption of the system, minimizing the average service delay, and balancing resource allocation. Based on the advantages of adaptive learning ability and sequence decision of DRL technology in the environment, DRL is introduced into the MEC system to solve the resource allocation problem and an intelligent

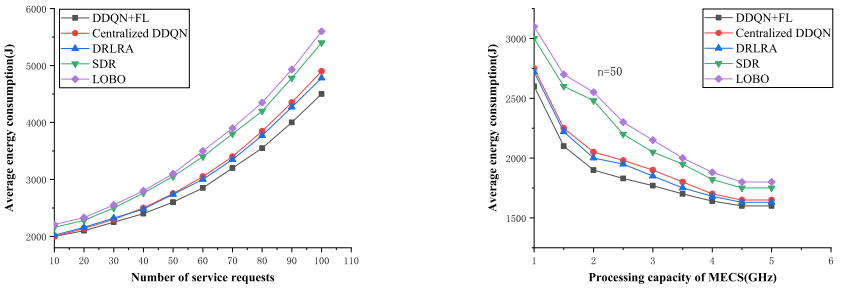

Fig. 6. Performance comparison of the average energy consumption for different resource allocation algorithms. (**a**) Comparison of the average energy consumption for different resource allocation algorithms with different number of service requests; (**b**) Comparison of the average energy consumption for different resource allocation algorithms with different processing capacities of the MEC servers.

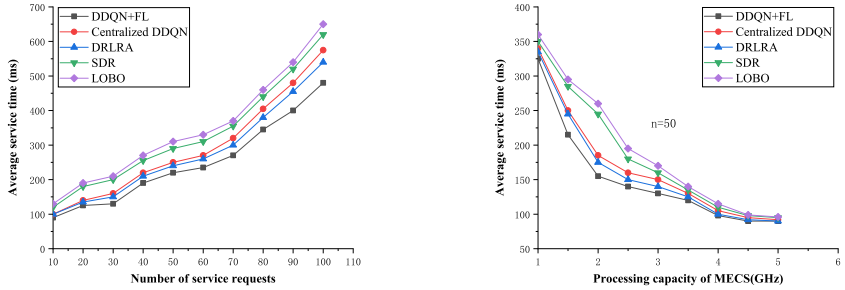

Fig. 7. Performance comparison of the average service time for different resource allocation algorithms. (**a**) Comparison of the average service time for different resource allocation algorithms with different number of service requests; (**b**) Comparison of the average service time for different resource allocation algorithms with different processing capacities of the MEC servers.

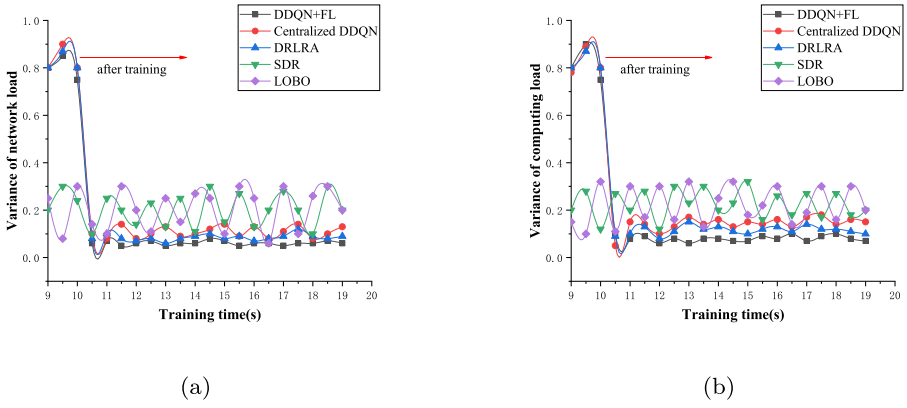

Fig. 8. The fluctuation of the load variance of different resource allocation algorithms. (**a**) The fluctuation of variance of the network load based on different resource allocation algorithms; (**b**) The fluctuation of variance of the computing load based on different resource allocation algorithms.

resource allocation algorithm DDQN-RA based on the emerging DRL algorithm framework DDQN is proposed. This algorithm can better adapt to changing mobile edge networks and is more suitable for MEC systems with changing network environments. In the subsequent deployment of DRL agents, we integrated the FL framework with DRL in the MEC system and proposed the "DRL + FL" model. This model can well solve the problems of uploading large amounts of training data via wireless channels, Non-IID and unbalance of training data when training DRL agents, restrictions on communication conditions, and data privacy. We conducted extensive simulation experiments to evaluate the performance of "DDQN + FL". The experimental results show that whether compared with traditional resource allocation algorithms SDR and LOBO or other intelligent resource allocation algorithms under multiple conditions, the "DDQN + FL" model proposed in this paper has achieved much better performance. Therefore, the combination of DRL techniques and FL framework has great potential in intelligent resource allocation, and it is worth further research. In our future work, we will not only optimize computing tasks in edge networks but also schedule

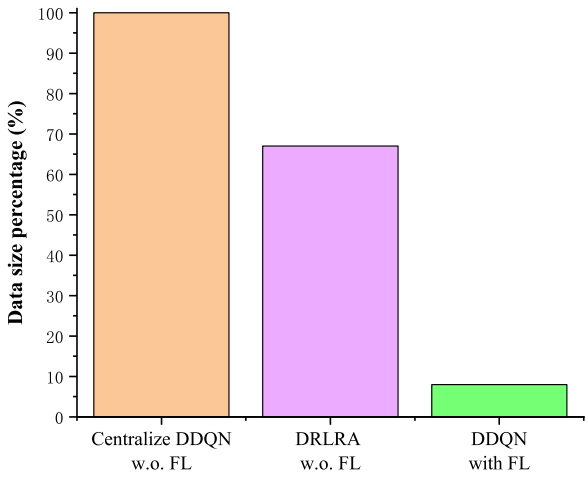

Fig. 9. Comparison of the transmission data size for different resource allocation algorithms.

artificial intelligence tasks on edge nodes and UMDs in a fine-grained and collaborative manner.

CRediT authorship contribution statement

Nanliang Shan: Conceptualization, Methodology, Visualization, Writing - original draft, Software, Writing - review & editing. **Xiaolong Cui:** Methodology, Supervision, Resources. **Zhiqiang Gao:** Conceptualization, Methodology, Visualization, Writing - review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Funding

This research was funded by the National Natural Science Foundation of China (Grant nos. U1603261) and the Natural Science Foundation Program of Xinjiang Province, China (Program No. 2016D01A080).

References

[1] M. Patel, B. Naughton, C. Chan, N. Sprecher, S. Abeta, A. Neal, et al., Mobile-edge computing introductory technical white paper, White paper, mobile-edge computing (MEC) industry initiative, 2014, pp. 1089–7801.
[2] A. Sadeghi, F. Sheikholeslami, G.B. Giannakis, Optimal and scalable caching for 5G using reinforcement learning of space-time popularities, IEEE J. Sel. Top. Sign. Proces. 12 (1) (2017) 180–190.
[3] Y. He, N. Zhao, H. Yin, Integrated networking, caching, and computing for connected vehicles: A deep reinforcement learning approach, IEEE Trans. Veh. Technol. 67 (1) (2017) 44–55.
[4] R.S. Sutton, A.G. Barto, Reinforcement Learning: An Introduction, MIT Press, 2018.
[5] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski, et al., Human-level control through deep reinforcement learning, Nature 518 (7540) (2015) 529.
[6] H.B. McMahan, E. Moore, D. Ramage, S. Hampson, et al., Communication-efficient learning of deep networks from decentralized data, 2016, arXiv preprint arXiv:1602.05629.
[7] Q. Li, Z. Wen, B. He, Federated learning systems: Vision, hype and reality for data privacy and protection, 2019, arXiv preprint arXiv:1907.09693.
[8] H. Van Hasselt, A. Guez, D. Silver, Deep reinforcement learning with double q-learning, in: Thirtieth AAAI Conference on Artificial Intelligence, 2016.
[9] M.-H. Chen, B. Liang, M. Dong, Joint offloading decision and resource allocation for multi-user multi-task mobile cloud, in: 2016 IEEE International Conference on Communications, ICC, IEEE, 2016, pp. 1–6.
[10] X. Chen, L. Jiao, W. Li, X. Fu, Efficient multi-user computation offloading for mobile-edge cloud computing, 2015, arXiv: Networking and Internet Architecture.
[11] Z.M. Fadlullah, F. Tang, B. Mao, N. Kato, O. Akashi, T. Inoue, K. Mizutani, State-of-the-art deep learning: Evolving machine intelligence toward tomorrow's intelligent network traffic control systems, IEEE Commun. Surv. Tutor. 19 (4) (2017) 2432–2455.
[12] J. Li, H. Gao, T. Lv, Y. Lu, Deep reinforcement learning based computation offloading and resource allocation for MEC, in: 2018 IEEE Wireless Communications and Networking Conference, WCNC, IEEE, 2018, pp. 1–6.
[13] Y. Liu, M.J. Lee, Y. Zheng, Adaptive multi-resource allocation for cloudlet-based mobile cloud computing system, IEEE Trans. Mob. Comput. 15 (10) (2015) 2398–2410.
[14] J. Wang, L. Zhao, J. Liu, N. Kato, Smart resource allocation for mobile edge computing: A deep reinforcement learning approach, IEEE Trans. Emerg. Top. Comput. (2019) 1–1.
[15] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, P. Bahl, MAUI: making smartphones last longer with code offload, 2010, pp. 49–62.
[16] L. Yang, J. Cao, Y. Yuan, T. Li, A. Han, A. Chan, A framework for partitioning and execution of data stream applications in mobile cloud computing, ACM SIGMETRICS Perform. Eval. Rev. 40 (4) (2013) 23–32.
[17] J. Evans, V. Krishnamurthy, Hidden Markov model state estimation with randomly delayed observations, IEEE Trans. Signal Process. 47 (8) (1999) 2157–2166.
[18] R.L. Dobrushin, Shannon's theorems for channels with synchronization errors, Problemy Peredachi Informatsii 3 (4) (1967) 18–36.
[19] Y. Wen, W. Zhang, H. Luo, Energy-optimal mobile application execution: Taming resource-poor mobile devices with cloud clones, in: Proceedings of the 2012 Proceedings IEEE Infocom, 2012, pp. 2716–2720.
[20] H. Van Hasselt, A. Guez, D. Silver, Deep reinforcement learning with double Q-learning, 2016, pp. 2094–2100.
[21] H. You, Z. Jiao, H. Xu, J. Li, Y. Wang, X. Gao, Restricting greed in training of generative adversarial network, 2017, arXiv: Computer Vision and Pattern Recognition.
[22] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (2015) 436–444.
[23] L. Chen, H. Qu, J. Zhao, B. Chen, J.C. Principe, Efficient and robust deep learning with correntropy-induced loss function, Neural Comput. Appl. 27 (4) (2016) 1019–1031.
[24] T.T. Anh, N.C. Luong, D. Niyato, Y.-C. Liang, D.I. Kim, Deep reinforcement learning for time scheduling in RF-powered backscatter cognitive radio networks, in: 2019 IEEE Wireless Communications and Networking Conference, WCNC, IEEE, 2019, pp. 1–7.
[25] Z. Hu, Y. Jiang, X. Ling, Q. Liu, Accurate Q-learning, in: International Conference on Neural Information Processing, Springer, 2018, pp. 560–570.
[26] H.B. McMahan, E. Moore, D. Ramage, S. Hampson, et al., Communication-efficient learning of deep networks from decentralized data, 2016, arXiv preprint arXiv:1602.05629.
[27] M. Lopezmartin, B. Carro, A. Sanchezesguevillas, Application of deep reinforcement learning to intrusion detection for supervised problems, Expert Syst. Appl. 141 (2020) 112963.
[28] G. Caminero, M. Lopezmartin, B. Carro, Adversarial environment reinforcement learning algorithm for intrusion detection, Comput. Netw. 159 (2019) 96–109.
[29] X. Wang, Y. Han, C. Wang, Q. Zhao, X. Chen, M. Chen, In-edge AI: Intelligentizing mobile edge computing, caching and communication by federated learning, IEEE Netw. 33 (5) (2019) 156–165.
[30] T. Soyata, R. Muraleedharan, C. Funai, M. Kwon, W. Heinzelman, Cloud-vision: Real-time face recognition using a mobile-cloudlet-cloud acceleration architecture, in: 2012 IEEE Symposium on Computers and Communications, ISCC, IEEE, 2012, pp. 000059–000066.
[31] B.P. Rimal, D.P. Van, M. Maier, Cloudlet enhanced fiber-wireless access networks for mobile-edge computing, IEEE Trans. Wireless Commun. 16 (6) (2017) 3601–3618.
[32] T.Q. Dinh, J. Tang, Q.D. La, T.Q.S. Quek, Offloading in mobile edge computing: Task allocation and computational frequency scaling, IEEE Trans. Commun. 65 (8) (2017) 3571–3584.
[33] S. Lakshminarayana, Y. Xu, H.V. Poor, T.Q.S. Quek, Cooperation of storage operation in a power network with renewable generation, IEEE Trans. Smart Grid 7 (4) (2016) 2108–2122.
[34] G. Chen, A new framework for multi-agent reinforcement learning–centralized training and exploration with decentralized execution via policy distillation, 2019, arXiv preprint arXiv:1910.09152.