Task Computation Offloading for Multi-Access Edge Computing via Attention Communication Deep Reinforcement Learning

Kexin Li, Xingwei Wang, Qiang He, Mingzhou Yang,
Min Huang, *Member, IEEE*, and Schahram Dustdar, *Fellow, IEEE*

Abstract—This article investigates how to enhance the Multi-access Edge Computing (MEC) systems performance with the aid of device-to-device (D2D) communication computation offloading. By adequately exploiting a novel computation offloading mechanism based on D2D collaboration, users can efficiently share computational resources with each other. However, it is challenging to distinguish valuable information that truly promotes a collaborative decision, as worthless information can hinder collaboration among users. In addition, the transmission of large volumes of information requires high bandwidth and incurs significant latency and computational complexity, resulting in unacceptable costs. In this article, we propose an efficient D2D-assisted MEC computation offloading framework based on Attention Communication Deep Reinforcement Learning (ACDRL), which simulates the interactions between related entities, including device-to-device collaboration in the horizontal and device-to-edge offloading in the vertical. Second, we developed a distributed cooperative reinforcement learning algorithm that includes an attention mechanism that skews computational resources towards active users to avoid unnecessary resource wastage in large-scale MEC systems. Finally, to improve the effectiveness and rationality of cooperation among users, we introduce a communication channel to integrate information from all users in a communication group, thus facilitating cooperative decision-making. The proposed framework is benchmarked, and the experimental results show that the proposed framework can effectively reduce latency and provide valuable insights for practical design compared to other baseline approaches.

Index Terms—Multi-access edge computing, reinforcement learning, task computation offloading, user cooperation

1 Introduction

The number of Internet of Things (IoT) devices and applications has increased dramatically and has opened a new era in which user equipment acts as both a data producer and consumer [1]. However, many user equipments can hardly afford the computing and memory resources demanded to process complex compute-insensitive tasks. Recently, Multi-access Edge Computing (MEC) technology has been proposed to solve this problem by distributing developing computation, storage, and other resources at edge nodes of the network, such as base station and access point [2]. The new computing paradigm brings computational resources closer to user equipment avoiding sending data to the network and therefore can support ultra-low latency and high computational applications. Furthermore, users can offload their computation-intensive tasks to the resource-rich remote clouds via wireless access. It gives rise to the challenging problem of how to optimize computation resources. Computation offloading service provides a feasible scheme for supporting MEC such that the demands of the optimization problems are met [3].

In the early stage of developing MEC computation offloading technology, some existing studies focus on minimizing the long-term time latency and energy consumption in a centralized manner [4], [5]. These approaches to coordination equipment need the global perspective to monitor the whole system status information, and obtain the offloading decision by centralized training. However, with the increase of intelligent devices and the emergency of computation applications, such as Virtual Reality/Augmented Reality (VR/AR), these approaches will increase the executing cost and communication cost, resulting in poor system performance. Meanwhile, the computing power of edge servers is usually limited, and these methods fluctuate seriously due to the influence of the network, so it may not be feasible to offload all tasks to edge

- *Kexin Li and Mingzhou Yang are with the College of Computer Science and Engineering, Northeastern University, Shenyang 110819, China. E-mail: neulikexin@stumail.neu.edu.cn, yangmingzhou11@163.com.*
- *Xingwei Wang is with the College of Computer Science and Engineering and State Key Laboratory of Synthetical Automation for Process Industries, Northeastern University, Shenyang 110819, China. E-mail: wangxw@mail.neu.edu.cn.*
- *Qiang He is with the College of Medicine and Biological Information Engineering, Northeastern University, Shenyang 110169, China. E-mail: heqiang@bmie.neu.edu.cn.*
- *Min Huang is with the College of Information Science and Engineering, Northeastern University, Shenyang 110819, China. E-mail: mhuang@mail.neu.edu.cn.*
- *Schahram Dustdar is with the Distributed Systems Group, TU Wien, 1040 Vienna, Austria. E-mail: dustdar@dsg.tuwien.ac.at.*

Manuscript received 18 July 2022; revised 27 October 2022; accepted 25 November 2022. Date of publication 25 January 2023; date of current version 8 August 2023.
This work was supported in part by the National Key R & D Program of China under Grant 2022YFB4500800; in part by the National Natural Science Foundation of China under Grants 62032013 and 61872073; in part by the LiaoNing Revitalization Talents Program under Grant XLYC1902010.
(Corresponding author: Xingwei Wang.)
Recommended for acceptance by F. Delicato.
Digital Object Identifier no. 10.1109/TSC.2022.3225473

servers [6]. Therefore, these centralized methods are not suitable for the dynamic MEC environment, especially those with delay-sensitive tasks.

A promising solution is to offload tasks through Devices-to-Devices (D2D) links by utilizing the computing resources of other user equipment, called D2D-assisted computation offloading scheme. It is shown in [7] that the D2D-assisted scheme can effectively improve the utilization of idle resources in the network, and at the same time, offloading tasks to other users proactively can substantially reduce the pressure on edge servers during peak-time traffic. Furthermore, a D2D-assisted scheme can coordinate channel interference in a time-varying communication environment and allow users to make intelligent offloading decisions in a distributed manner. Specifically, in case of multiple user equipment request computing services from collaborative computing between users instead of the same edge server.

Most existing D2D-assisted edge computation offloading schemes focus on the optimization or a game-theoretical method, in which computation offloading and resource allocation problems are combined and formulated as mixed-integer linear programming problems [8], [9], [10], [11], [12]. Besides, most of the literature usually solves the problem within a time slot, e.g., only focusing on the channel conditions within a time slot. However, they have to update the channel conditions and resolve the offloading and resource allocation problems when the channel conditions change. It brings enormous computational overhead to finding optimal decisions, especially for dynamic decision-making problems like the Internet of Vehicles (IoV). Existing methods are time-consuming and no longer suitable for dynamic environment. Although some studies have attempted to model the problem as a Markovian problem, the explosion of dimensionality and indeterminate transition probabilities increase the complexity of optimization. Meanwhile, Multi-agent Deep Reinforcement Learning (MADRL) has attracted much attention as the most popular machine learning technique, such as Multi-Agent Deep Deterministic Policy Gradient (MADDPG) [13], Distributed Multi-Agent Policy Gradient (DiMA-PG) [14]. In addition, high-performance computer technology enables MADRL to realize the ability to handle dynamic and complex tasks [15], [16], [17]. Several pioneering research works have employed MADRL techniques to address D2D-assisted computational offloading, as it excels in coping with dynamic environments and making sequential decisions under the uncertainty of multiple agents [18], [19], [20], [21]. Although these methods can well capture the features of mixed cooperative-competitive relationships among users and are highly robust and scalable, all the above works are based on the predefined communication of the D2D-assisted architecture and assume that all users share global information. In a dynamic user environment, the predefined D2D-assisted architecture cannot distinguish the valuable information that facilitates collaborative decision-making from a large amount of globally shared information. In this case, communication cannot be helpful and can hinder user collaboration. In addition, transferring large amounts of information requires high communication costs and can incur significant system latency and computational complexity.

Inspired by the above research, we propose an Attentions Communication Deep Reinforcement Learning (ACDRL) algorithm, an innovative paradigm focused on computational offloading that can handle dynamic D2D-assisted communication architectures, especially under a partially observable environment in a dynamic user environment. First, this system model considers inseparable tasks and builds a D2D-assisted horizontal model to simulate the relationship between different user entities, including horizontal D2D communication among users and vertical communication between users and edge nodes. A queue system is applied to estimate the offloading process of tasks. Unlike the predefined D2D-assisted communication architecture in previous work, the state of the D2D-assisted architecture is timely changing in this system. Second, for ACDRL, an attention mechanism is proposed to generate the thoughts of users based on the current states, further reflecting whether the user needs computation services or not. If so, called initiator, then selected other users, called collaborators, and form a communication group. In practice, the states of a user (initiator) and its communication group will change due to the state of tasks. Therefore, each initiator needs to grasp the states of collaborators in the communication group at each time slot. Moreover, we introduce a bidirectional Long Short-Term Memory (LSTM) unit as the communication channel to connect each user within a communication group and integrate the action intents of whole users. Different from the works as mentioned in [22], the LSTM unit in this system takes internal states as input and returns thoughts that guide users for coordinated communication group strategies. It can take advantage of its structure to select suitable users within its observations and outputs important information for cooperative decision-making, making it possible to learn coordinated strategies in dynamic communication environments. This process can reduce the dimension of the D2D-assisted computation offloading group, reducing the dimension of the state observed by ACDRL. Finally, according to the states information of other users in the communication group, the actor-critic-based model is used to train the new experience replay with communication group information and obtain the offloading decisions.

In this framework, we use D2D-link to offload tasks to other User Equipment (UE) with idle computing resources, which improves the utilization of computing resources in the MEC system. At the same time, by taking advantage of the short distance between UEs, the communication cost is effectively reduced, thereby reducing the system cost. In addition, our proposed architecture only focuses on the UEs that require computation offloading services (initiator) and have idle computing resources (collaborator), which can significantly reduce the waste of computing resources, especially in dynamic scenarios. On the other hand, our offloading decision-making system is arranged on each UE. The D2D-assisted MEC computation offloading system can realize information sharing and promote cooperation among UEs, and optimize the computation offloading decision-making process in dynamic scenarios. The major contributions of this article are summarized as follows:

1) *D2D-assisted Computation Offloading Architecture in MEC system.* We propose a novel D2D-assisted edge computation offloading architecture that includes both horizontal and vertical cooperation in dynamic

users MEC systems. It enables users to simultaneously offload their tasks to edge nodes and other users with idle computing resources in the communication range and makes efficient assignment decisions online.

2) *ACDRL-based Computation Offloading Model.* We formulate the tasks computation offloading as a Markov Decision Process (MDP) minimization problem and propose a cooperative multi-agent reinforcement learning ACDRL to minimize the average system tasks delay. Specifically, when fixing offloading decisions, we can reset the settings of the UE and its communication group based on the latest system information. Based on this, we adopt a task offloading algorithm to give an effective dynamic task offloading scheme.
3) *Performance Evaluation.* We conduce extensive experiments to evaluate the performance of ACDRL scheme. The simulation results demonstrate the efficacy of our proposed algorithm compared to existing schemes can reduce 12.7% of the system latency cost and improve 20.7% task completion rate in the system. And provide insights related to the cooperative computation offloading in dynamic user MEC environment.

The rest of this article is organized as follows. In Section 2, the related work is described. Section 3 details the offloading problem as a partial observable MDP. In Section 4, the cooperative decision-making problem among independent multiusers is proposed, and an ACDRL scheme is derived to solve the problem in Section 3. The simulation results are presented in Section 5. Finally, Section 6 summarizes this article.

2 Related Work

Computation offloading in MEC has been widely regarded as a pivotal technology to improve the Quality of Service (QoS) performance of applications, which are the critical and fundamental parts of D2D communications. Depending on their execution modes, existing D2D-assisted computation offloading methods can be divided into centralized and decentralized schemes. In the centralized schemes, the edge node is responsible for computing resources for the users and monitoring information such as the interference level of users and network states. For instance, Yang et al. [9] proposed an offloading method by regarding the computation offloading process as a resource contention game, which minimized the individual task execution cost and the system overhead. Lan et al. [23] assumed that the task offloading and the resource optimization problem are formulated as a mixed-integer nonlinear programming problem (MINLP) with joint consideration of the user equipment allocation policy, task offloading policy, and computational resource allocation policy to find the optimal offloading strategy in D2D systems. All these works targeted at deriving the optimal strategy for one time slice under an optimization or game framework, while not considering the dynamic channel conditions time-varying environments of the network. Furthermore, some Deep Reinforcement Learning (DRL) algorithms have been developed to solve the computation offloading problem in dynamic communication environments, especially in dynamic user scenarios [24], [25], [26], [27]. However, centralized schemes require edge nodes to have global information. In the dynamic network environment, such as dynamic user requests and network dynamics, the complexity of the centralized schemes increases with the number of users, resulting in enormous traffic and computation pressure on the edge nodes.

To reduce the computation load of edge nodes and the communication load of traffic, a series of distributed approaches based on D2D communications schemes in which users make offloading decisions independently. For instance, Chen et al. [28] proposed a D2D-enable multihelper MEC system and proved that D2D-assisted computational offloading communication mode could offload wireless traffic from wireless network infrastructure, which was considered as the most promising distributed collaborative computing offloading technology in MEC system. Xing et al. [29] investigated the joint task assignment and communication rate for D2D-enabled multi-helper MEC system assuming binary task offloading. This work enhanced the local user's computation latency by exploiting D2D collaborations at the network edge. However, it is not practiced since they assumed that all users are deployed at fixed locations with static wireless channels. Lin et al. [30] investigated computation offloading designs for D2D cooperative between the two users who can dynamically exchange the computation loads via the D2D links. Although this design realized minimization optimal within a given finite time horizon, it ignored the scalable and efficient cooperation.

A few recent studies have attempted to adopt the MADRL algorithm to solve D2D-assisted computation offloading optimization problems. For instance, Shi et al. [20] considered a specific Stackelberg game solution and adopted neural networks as the functional approximator in the MADRL, reducing the time cost in the negotiation process and achieving time-efficient resource allocation. Chen et al. [31] formulated the computation offloading problem as a multi-agent Markov decision process, for which a distributed learning framework was proposed beyond fifth-generation networks. Moreover, the rapid development of the MADRL algorithm increases by jointly considering it and other optimal methods, such as game theory and federated learning, to achieve excellent performance in the MEC computation offloading [32], [33]. Unlike the existing frameworks that predefined D2D communication and cannot handle dynamic users scenario, we consider a dynamic D2D-assisted computation offloading architecture for efficient utilization of computing resources. Moreover, we propose an ACDRL algorithm to address the problems of complicated convergence and ineffective cooperation brought by dynamic D2D communication. It further reduces the system latency and achieves time-efficient computation offloading in the dynamic user MEC scenario.

3 System Model and Assumptions

In this section, we introduce the system modeling of D2D-assisted collaborative edge computation offloading. Particularly, we present the corresponding network architecture in Sections 3.1, and 3.2 introduces the system model, including local computing, edge computing, and D2D-assisted computing architecture. Section 3.3 models the problem formulation.

TABLE 1
Default Simulation Parameters

Parameter	Description
M^p	Number of UE
M	Number of initiator
N	Number of EN
I	Number of collaborator
δ_m^t	Max tolerance latency of task x_m^t
c_m^t	The data size of task x_m^t
z_m^t	Requested CPU of the task x_m^t
ω_m^t	Max waiting delay of the task x_m^t
$\mathbf{D}_m^t$	Total delay of task x_m^t
g_m	Computing capacity of initiator m
f_n	Computing capacity of EN n
σ_n^t	Wait time at the queue of EN n
$b_{m,n}^{e,t}$	Channel bandwidth between initiator m and EN n
$\varrho_{m,n}^t$	Transmitting power between initiator m and EN n
$h_{m,n}^t$	Channel gain between initiator m and EN n
L	Path loss at a unit distance
$d_{m,n}^t$	Distance between initiator m and EN n
ς^2	Gaussian noise ratio
$b_{m,i}^{u,t}$	Channel bandwidth between initiator m and collaborator i
$\varrho_{m,i}^t$	Transmitting power between initiator m and collaborator i
$h_{m,i}^t$	Channel gain between initiator m and collaborator i
q_i	Computing capacity of collaborator i
λ_i^t	Maximum amount time of collaborator i
$r_{m,n}^t$	Transmission rate between initiator m and EN n
$r_{m,i}^t$	Transmission rate between initiator m and collaborator i
$\mathcal{O}_m^t$	Indicator of the task need offload or not
$\mathcal{Y}_{m,n}^t$	Indicator of the task offload to EN n or not
$\mathcal{Z}_{m,i}^t$	Indicator of the task offload to collaborator i or not
γ^t	Discount factor
$\mathcal{H}_m^t$	Thought of initiator m at time slot t
$\tilde{\mathcal{H}}_m^t$	Integrated thought of communication group of initiator m at time slot t

Some key modeling parameters and notations are summarized in Table 1.

3.1 Network Architecture

An illustration of the network architecture of D2D-assisted collaborative edge computation offloading is shown in Fig. 1. In this system, the operational timeline in our system is discredited into time slot $\mathcal{T} = \{1, 2, \dots t\}$, where each time slot has a during of Δ seconds. The duration matches the timescale at which task offloading could be updated [34]. The network architecture of D2D-assisted consists two entitles: The Edge Nodes (ENs) and UEs.

EN: We consider a fundamental model consisting of a set $N = \{1, 2, \dots n\}$ of ENs. In this MEC system, we assume that each EN deploys a Base station (BS) and an edge server (ES); BS is mainly used for communication while ES provides computing services and each BS has a stable power supply

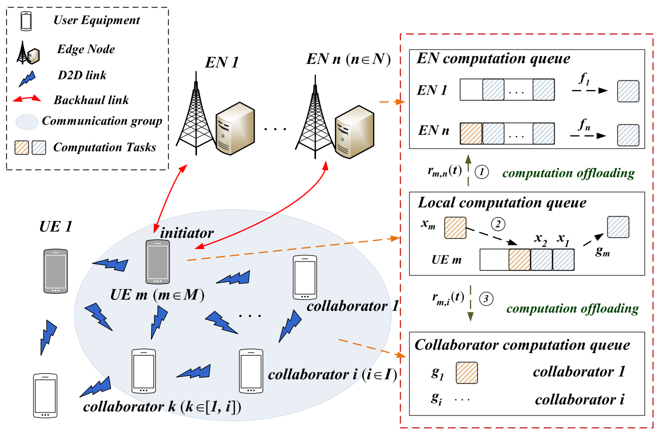


Fig. 1. System model of user cooperation MEC.

and can broadcast radio frequency energy to ES. Each ES has more excellent computation capacity than UEs. The computation capability and the bandwidth resources of EN m are defined as f_m (cycles per second) and B_m, respectively.

UE: Let $M^p = \{1, 2, \dots m^p\}$ denote the set of UEs. In practice, UEs may correspond to smart devices or a low-power IoT system, i.e., Smart Home, Autonomous Driving. In addition, we introduce the concepts of *initiator*, *collaborator*, and *communication group*. Depending on the task's requirements and the equipment's computing capacity, UEs are tagged as initiator and collaborator. Initiator refers to the UE who needs the computation offloading service. The Initiator selects other users with idle computation resources within D2D links communication, called collaborators, and in each Initiator's field to form a communication group for coordinated strategies. A set of initiators $M = \{1, 2, \dots m\}$, and a set of collaborators $I = \{1, 2, \dots i\}$, where $M \subset M^p, I \subset M^p$. Each Initiator is assumed to have a finite computation capability g_m (cycles per second). It connects to EN by Wireless Fidelity and other collaborators by D2D links. We focus on the computational tasks x_m^t of initiator m at time slot t, which can be processed depending on computation offloading decision in three cases: 1) locally, 2) offloading to EN n, and 3) offloading to other collaborators which would like to share the idle computation resources. For any initiator, a new task arrival probability with Poisson distribution ϑ^t at the start of each time interval. We denote the task of initiator m as $x_m^t = \{c_m^t, z_m^t, \delta_m^t, g_m\}$, where c_m^t, z_m^t, δ_m^t, and g_m are defined as the data size, the requested CPU cycles, the max tolerance latency of the task, and the computing capacity of initiator m, respectively.

At the beginning of each time slot t, a new task arrives at initiator m, the attention mechanism will determine whether the user is an initiator. Then, the communication channel will select I collaborators to form a communication group C. Second, the agent selects one collaborator of the communication group or EN n to execute the task according to the computation offloading decision. Attention mechanisms and communication channels will be described in detail in Sections 4.2 and 4.3. Eventually, the agent collects the global information, including data sizes, network status, the processing capacity of initiator m, collaborator i, and EN n, then it makes an offloading decision according to this information.

3.2 System Model

3.2.1 *Local Computing*

In this article, we assume that each computation task is indivisible. Considering the local computing processing, the generated task of initiator m is sent to the computing queue of the local device. The time consumption of the task x_m^t depends on the computation capacity g_m of the initiator m, the required CPU cycles z_m^t. Thus, the local delay $D_m^l(t)$ can be expressed as $D_m^l(t) = z_m^t/g_m$. We can obtain the energy consumption of task x_m^t for local computing processing as $E_m^l(t) = \rho_m \cdot z_m^t$, where ρ_m is the power coefficient of energy consumed per CPU cycle at initiator m.

3.2.2 *Edge Computing*

Next, we analyze the delay of edge computing. To execute tasks at EN n, initiator m first needs to transmit the input data with the size of c_m^t to the EN n. Similar to many studies, the returning data is generally much smaller than the input data [35]. Thus, we assume that the transmission time of return delay can be ignored. Consequently, if the offloading strategy of initiator m is offloading the task to edge server n, then we will consider the transmission delay $D_{m,n}^{tr}(t)$, waiting delay σ_n^t at queue of EN n, and execution delay $D_{m,n}^{ex}(t)$. On one hand, the data transmission delay is determined by the data size c_m^t, the distance $d_{m,n}^t$ and the transmission rate $r_{m,n}^t$. In the communication model of this system, the EN can serve as a multiple access scenario. As a realistic approach, we assume that EN n owns bandwidth resources that can be divided into orthogonal sub-channels of size $b_{m,n}^{e,t}$ Hz each according to [36]. Therefore, we define $W = \{1, \cdots, w\}$ as the set of available sub-channels for initiator. Each sub-channel can be allocated to at most one initiator. At time slot t, the bandwidth allocated to initiator m in the form of EN n is denoted as $b_{m,n}^{e,t} = B_m \frac{P_{m,n}^t}{\sum_W P_{m,w}^t}, m \in M, n \in N$, where $P_{m,n}^t$ is a set of bandwidth allocation factors based on realistic conditions. When $P_{m,n}^t = 1$, the communication resources of EN n are equally allocated to initiators. In terms of the Shannon Theorem [37], the transmission rate $r_{m,n}^t$ from initiator m to EN n can be calculated by

$$r_{m,n}^t = b_{m,n}^{e,t} \log_2\left(1 + \frac{\varrho_{m,n}^t h_{m,n}^t}{L * d_{m,n}^t + \varsigma^2}\right), \tag{1}$$

where $\varrho_{m,n}^t$ is the transmitting power, $h_{m,n}^t$ is the antenna gain at EN n, L is the path loss at a unit distance and $d_{m,n}^t$ is the distance between initiator m and EN n. ς^2 is the power of additive white Gaussian noise of edge computation offloading.

Consequently, if the computation task is executed at EN n, the transmission delay $D_{m,n}^{tr}(t)$ of computation task x_m^t can be expressed as $D_{m,n}^{tr}(t) = c_m^t/r_{m,n}^t$. If the computational task x_m^t is selected as offloading to EN n, the execution delay $D_{m,n}^{ex}(t)$ can be given as $D_{m,n}^{ex}(t) = z_m^t/f_n$, where f_n represents the CPU clock speed of the server at EN n. Although the communication delay of EN is affected by distance, dynamic bandwidth, and channels, $D_{m,n}^{ex}(t) < D_m^l(t)$ generally holds in practical systems [32]. Furthermore, it should be noted that in this system, taking the waiting delay into account for the execution delay is necessary for EN offloading scenarios. Thus, the total delay of edge computing equals the execution delay pulses the transmission delay and waiting time of EN n which can be expressed as $D_m^e(t) = D_{m,n}^{tr}(t) + D_{m,n}^{ex}(t) + \sigma_n^t$.

Furthermore, we can obtain the transmission energy consumption for initiator offloading the task x_m^t to EN n is $e_{m,n}^{tr}(t) = c_m^t \cdot \varrho_{m,n}^t/r_{m,n}^t$. And obtain the execution consumption for edge computing processing at EN n as $e_{m,n}^{ex}(t) = \rho_n \cdot z_m^t$, where ρ_n is the power coefficient of energy consumed per CPU cycle at EN n. Therefor, we can obtain the total energy consumption $E_m^e(t)$ for edge computing offloading the task x_m^t to EN n: $E_m^e(t) = e_{m,n}^{tr}(t) + e_{m,n}^{ex}(t)$.

3.2.3 *D2D-Assisted Computing*

If the request cannot be satisfied through the local computing, initiator also can seek help from surrounding users that would like share idle computation resources (we assume that every UE can establish direct D2D links), called D2D-assisted computation offloading model. We define $i \in I$ and $\mathrm{SNR}_{m,i}^t$ as the index for communication group of initiator m and the signal-noise-ratio (SNR) at collaborator i assigns channel to initiator m, respectively. Therefore, the $\mathrm{SNR}_{m,i}^t$ can be formulated as follow:

$$\mathrm{SNR}_{m,i}^t = \frac{|(1 + \varrho_{m,i}^t |h_{m,i}^t|^2))\rho_m|^2}{\sigma_i^2}, i \in M, m \in M, \tag{2}$$

where $\varrho_{m,i}^t$ and $h_{m,i}^t$ denote the corresponding transmit power, and a ratio function of the channel gain, respectively. $(\rho_m)^2$ is the transmit power of initiator m, σ_i^2 denotes the additive white Gaussian noise at the collaborator i. Then, the corresponding transmission rate among users by D2D links can be calculated as

$$r_{m,i}^t = b_{m,i}^{u,t} log_2(1 + \mathrm{SNR}_{m,i}^t). \tag{3}$$

Therefore, the D2D-assisted computation offloading transmission delay $D_{m,i}^{tr}(t)$ for task x_m^t of initiator m to fetch the content through D2D communication can be calculated as $D_{m,i}^{tr}(t) = c_m^t/r_{m,i}^t$, where c_m^t is the size of content x_m^t. If the computation task will be offloaded to collaborator i, the execution delay can be given as $D_{m,i}^{ex}(t) = z_m^t/q_i$, where q_i represents the CPU clock speed of the server at collaborator i, thus the total delay equals the execution delay adding the transmission delay which can be expressed as $D_m^u(t) = D_{m,i}^{tr}(t) + D_{m,i}^{ex}(t)$. Unlike the Edge computing considering the waiting delay, we assume that only idle UEs can share their resources, so there is no waiting delay in the D2D-assisted computation offloading scheme. In addition, we set a maximum amount of time λ_i^t for each collaborator to ensure their tasks can be executed timely, which means that the computation tasks need to be completed within λ_i^t for offloading to collaborator i.

Similarly to edge computing, we can obtain that the total energy consumption for D2D-assisted computing, the total energy consumption for offloading the task x_m^t to collaborator i is $E_m^u(t) = e_{m,i}^{tr}(t) + e_{m,i}^{ex}(t)$, where $e_{m,i}^{tr}(t) = c_m^t \cdot \varrho_{m,i}^t/r_{m,i}^t$, $e_{m,i}^{ex}(t) = \rho_i \cdot z_m^t$, and ρ_i is the power coefficient of energy consumed per CPU cycle at collaborator i.

3.3 Problem Formulation

The offloading framework makes decisions by minimizing the long-term delay which interacts with the MEC environment. For each task generated by initiator, we denote set $\{\mathcal{O}_m^t, \mathcal{Y}_{m,n}^t, \mathcal{Z}_{m,i}^t\}$ as the offloading status of task x_m^t, where $\mathcal{O}_m^t, \mathcal{Y}_{m,n}^t, \mathcal{Z}_{m,i}^t \in \{0,1\}, \mathcal{O}_m^t + \mathcal{Y}_{m,n}^t + \mathcal{Z}_{m,i}^t = 1$. $\mathcal{O}_m^t = 1$ means the task will be executed locally, $\mathcal{O}_m^t = 0$ means the task will be executed at EN or collaborator. And $\mathcal{Y}_{m,n}^t = 1$ indicates whether EN n is selected to perform the computation offloaded task, $\mathcal{Z}_{m,i}^t$ indicates whether collaborator i is selected to perform the task; otherwise, $\mathcal{Z}_{m,i}^t = 0$. Therefore, for a task generated by initiator m, the total task delay can be denoted as

$$\mathbf{D}_m^t = \mathcal{O}_m^t \cdot D_m^l(t) + \mathcal{Y}_{m,n}^t \cdot D_m^e(t) + \mathcal{Z}_{m,i}^t \cdot D_m^u(t). \tag{4}$$

Note that we try to make a offloading decision at each time slot to minimize the long-term process time of tasks with some constrains, i.e., the total energy consumption is under the energy budget E_{budget}. Therefore, in this system, the task offloading problem can be formulated as follows:

$$\text{Minimize} \sum_{m=1}^{M} \sum_{t=1}^{\mathcal{T}} \mathbf{D}_m^t, \forall m \in M, \forall t \in \mathcal{T} \tag{5a}$$

$$\text{s.t. } C1: \mathcal{O}_m^t, \mathcal{Y}_{m,n}^t, \mathcal{Z}_{m,i}^t \in \{0,1\}, \forall m \in M, \forall t \in \mathcal{T} \tag{5b}$$

$$C2: D_m^u(t) < \lambda_i^t, \forall m \in M, \forall i \in \mathcal{I} \tag{5c}$$

$$C3: D_m^e(t) < \sigma_n^t, \forall m \in M, \forall n \in N \tag{5d}$$

$$C4: D_m^e(t), D_m^u(t) < \delta_m, \omega_m^t, \forall m \in M \tag{5e}$$

$$C5: E_m^l(t), E_m^e(t), E_m^u(t) < E_{budget}, \forall m \in M. \tag{5f}$$

It is rather challenging to achieve the above objective since the optimization problem in (5) is NP-Hard. Next, the proof of NP-Hardness for this problem is presented.

Theorem 1. *The optimization problem is NP-Hard.*

Proof. Consider there are M tasks $x_1^t, x_2^t, \ldots, x_m^t$ with the offloading energy budget E_{budget}, which means that with different offloading schemes for each time slot, the total energy consumption cannot exceed the energy budget for each task. We use D_m to denote total system latency of task x_m^t, and E_m to denote total energy consumption for task x_m^t. Then, our optimization problem can be formulated as

$$\text{Minimize} \frac{\sum_{m=1}^{m=M} D_m}{M} \tag{6a}$$

$$\text{s.t.} \sum_{m \in M} E_m \leq E_{budget}. \tag{6b}$$

□

We introduce the trivial 0-1 knapsack problem [38]: "maximize: $\sum_{i=1}^{n} = \omega_i \cdot x_i$, subject to: $\sum_{i=1}^{n} v_i \cdot x_i \leq C, x_i \in \{0,1\}$." Here, ω_i and v_i denote the weight and volume of the ith item, and C means the capacity of the knapsack. By mapping w_i, v_i, and C in the trivial 0-1 knapsack problem to D_m, E_m, and E_{budget} in the optimization problem of this computation offloading scheme, we get the two problems to be equivalent. Therefore, the optimization problem of this computation offloading scheme is NP-Hard. This completes the proof.

As Theorem 1 proved, the problem in (5) is NP-Hard, and the computation offloading decision is memoryless with a sequential decision-making process. Therefore, we formulate the task computation offloading as an MDP minimization problem. Nevertheless, traditional approaches have extremely high computation complexity, which may have some limitations in practical applications, especially in a dynamic network. At the same time, the D2D-assisted communication group dynamically changes. The previous work in designing the D2D-assisted offloading model usually assumes that the communication group is fixed and can not react to the D2D-assisted communication group changes due to their poor cooperatively. Therefore, it is necessary to introduce more intelligent methods in the system.

4 Optimization Algorithm

To deal with the above problem, we propose an Attention Communication Deep Reinforcement Learning (ACDRL) framework for the MEC computation offloading including three main phases, i.e., local Actor-Critic-based model training, attention mechanism, and communication channel. Particularly, the three basic elements of MDP are introduced in Section 4.1, and the details of the training and the whole flow in the processed ACDRL framework are introduced in Section 4.2. The learning framework of ACDRL for the computation offloading is depicted in Fig. 2.

4.1 Three Basic Elements

The offloading framework makes decisions by searching the policy strategy π for each agent and minimizing the long-term reward (system latency) $\mathbf{r}_t$ which interacts with the MEC environment. The policy π is a mapping from its states $\mathbf{s}_t$ to its action $\mathbf{a}_t$. In this system, the objective is to find an optimal policy to optimize the expected long-term system latency.

4.1.1 UE State

At the beginning of time t, each UE observes the global state information, including the task state, UE state, EN state, D2D link rate, and the network state from initiator to EN. We define the system states as $\mathbf{s}_t = \{\mathcal{X}_t, \mathcal{C}_t, \mathcal{E}_t, r_t^D, r_t^N\}$, where $\mathcal{X}_t = \{x_1^t, x_2^t, \ldots x_m^t\}$ denotes the task state of each UE at time slot t. $\mathcal{C}_t = \{c_1, c_2, \ldots c_i\}$ denotes the state of collaborators of communication group, where $c_i = \{\lambda_t, \mathcal{Q}_t\}$, $\lambda_t = \{\lambda_1^t, \lambda_2^t, \ldots \lambda_i^t\}$ denotes the maximum amount of time that can be occupied of collaborator i at time t, and $\mathcal{Q}_t = \{q_1, q_2, \ldots q_i\}$ denotes the computation capability of collaborators. $\mathcal{E}_t = \{\mathcal{F}, \sigma_t\}$ denotes the states of ENs, where $\mathcal{F} = \{f_1, f_2, \ldots f_n\}$ denotes the computation capability of ENs, and $\sigma_t = \{\sigma_1^t, \sigma_2^t, \ldots \sigma_n^t\}$ denotes the waiting time of EN n at time slot t. $r_t^D = \{r_{m,1}^t, r_{m,2}^t, \ldots r_{m,i}^t\}$ denotes the D2D link rate between initiator m to collaborator i. $r_t^N = \{r_{m,1}^t, r_{m,2}^t \ldots r_{m,n}^t\}$ denotes the link rate at time slot t from initiator m to EN n.

4.1.2 UE Action

After receiving the state $\mathbf{s}_t$, initiator m will decide where to offload the task through above methods. The action of initiator m at time slot t can be defined as $\mathbf{a}_t = \{\mathcal{O}_m, \mathcal{Y}_{m,n}^t, \mathcal{Z}_{m,i}^t\} \in$

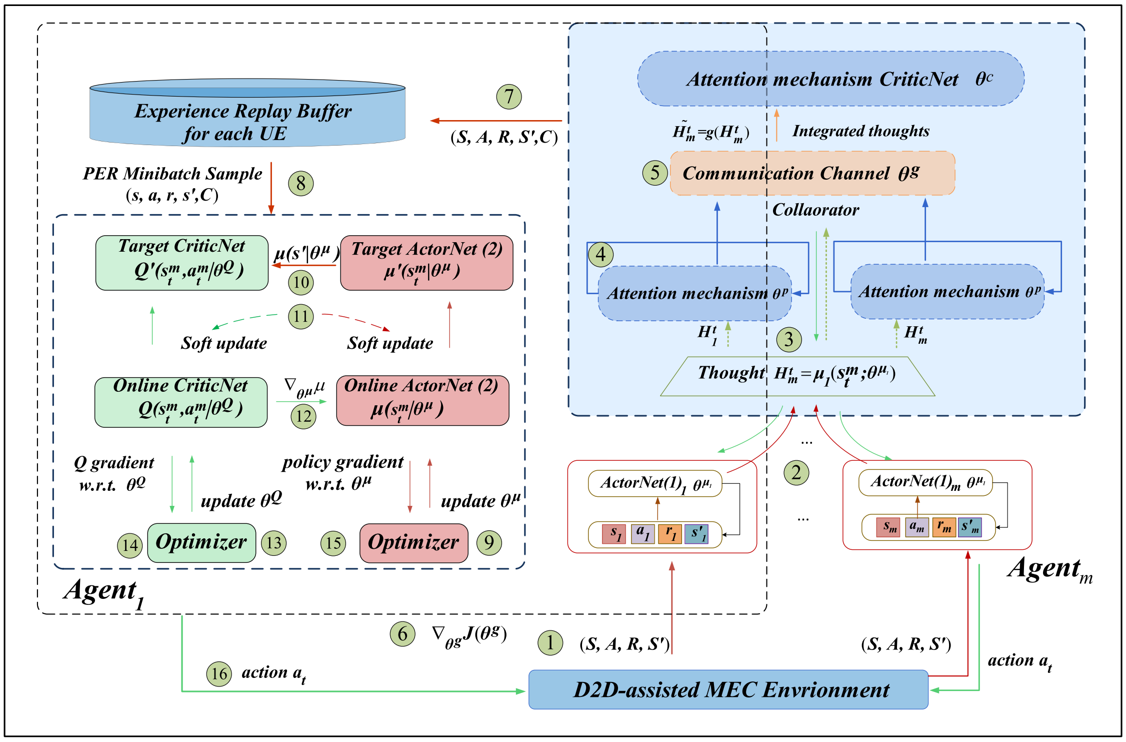


Fig. 2. System model of user cooperation MEC based on ACDRL.

$\{0,1\}$, where $\mathcal{O}_m^t + \mathcal{Y}_{m,n}^t + \mathcal{Z}_{m,i}^t = 1$. That is, the task computation offloading decision should be learned.

4.1.3 System Reward

The agent interacts with the environment by finding the corresponding relationship between state $\mathbf{s}_t$ and action $\mathbf{a}_t$, and guides the selection of the next action according to the reward $\mathbf{R}_t$ of each step. To reduce the system latency of the computation tasks, we consider the system latency in this computation offloading process as the system reward.

In order to derive the reward function of this model, we find an immediate utility at time t to quantify the task computation reward for the tasks of UEs. Therefore, in this system, we define the system reward of execution as $\mathbf{R}_t$ which can be expressed as $\mathbf{R}_t = \sum_{m=1,t=1}^{M,\mathcal{T}} \gamma^t \cdot \mathbf{r}(\mathbf{s}_t, \mathbf{a}_t), \forall m \in M, \forall t \in \mathcal{T}$, where the discount factor $\gamma \in [0,1]$ demonstrates the importance of the instant reward. And $\mathbf{r}(\mathbf{s}_t, \mathbf{a}_t) = \mathbf{D}_m^t$ at each time slot t.

4.2 Offloading Method Based on ACDRL Algorithm

4.2.1 The Whole Process

We consider each UE as an agent and possesses three types of components: Attention mechanism, Communication channel, and Actor-Critic network. First, for each UE agent in this system, we consider the partially observable distributed environments where each UE agent m (the agent on initiator m) receives local state $\mathbf{s}_t^m$ at each time slot $t, t \in \mathcal{T}$ (process 1 of Fig. 2), the ActorNet(1) network takes local states as input and extracts a hidden layer as *thought* which encodes local states and action intentions, represented as $\mathcal{H}_m^t = \mu_1(\mathbf{s}_t^m; \theta^{\mu_1})$ (process 2 of Fig. 2). Within each episode $\mathcal{T}$, the attention mechanism takes *thought* as input and decides whether that UE needs to cooperate or not (processes 3 and 4 of Fig. 2). If needed, it is called the initiator and selects other UEs called collaborators to form a communication group as we just mentioned at Section 3.1.

At the same time, the communication group is determined by the attention mechanism and maintained the same within episode $\mathcal{T}$. The communication channel connects with other agents in the communication group of initiator m, inputs the *thoughts* of other agents (local states and action intention), and outputs the integrated thoughts to guide the next action intention of the agent (process 5 of Fig. 2). By sharing local observational information and encoding action intent in a dynamic group, the respective agents can establish an association of global environmental predictions, guide other agents, and cooperate to make action intention decisions.

Finally, the generation of the offloading action relies on the use of the Deep Neural Network (DNN). It considers the historical transition as the input, and outputs approximate $Q(\mathbf{s}_t, \mathbf{a}_t)$. The agent can observe the state $\mathbf{s}_t$ from the environment, choose a course of action $\mathbf{a}_t$ according to the policy $\pi(\mathbf{s}_t, \mathbf{a}_t)$, and generate an immediate reward $\mathbf{r}_t$ (process 6 to 16 of Fig. 2).

4.2.2 Attention Mechanism

In the study of distributed multi-agent reinforcement learning algorithms, making a final decision requires that the state of all agents be fully considered [39]. Traditional deep reinforcement learning cannot solve the problems caused by high-dimensional state space, and dynamic networks increase the complexity of state space further. However, in MEC computation offloading schemes, the states of users are affected by task arrival rate, computing power, network, and other dynamic conditions. Therefore, offloading services are not necessary for each user. Mainly based on the D2D-assisted computation offloading architecture, users may constantly switch between the identities of 'need help' and 'help'. Inspired by the attention mechanism [40], we only focus on the user who requires offloading services, and other users as a member of the communication group only requires basic information, which can significantly reduce the complexity of the calculation while reducing the delay of the offloading process, and further improving the computation offloading efficiency and effective.

Therefore, our attention mechanism cannot observe global information in total, but can encode observable field and action intention to determine whether the agent requires offloading services, in other words, whether it wants to be an initiator. Specifically, the attention mechanism is an RNN network, at every fixed time step, the hidden state of the previous step and the encoding including the local observations and actions of the agent are defined as an input at this step, and the output is a classifier to determine whether it is an initiator. In addition, it does not need attention at every time step to determine whether the user wants to be the initiator, because collaboration strategies take a while to be effective. We usually set a fixed time to update initiators and communication groups in a specific scenario. The pseudo-code of the method is shown in Algorithm 1.

4.2.3 Communication Channel

The previous studies of D2D-assisted model architecture always fixed the system model initially, which made the computation offloading process substantial computational complexity. As we mentioned earlier, reducing the computation dimension by keeping active collaborators retained in the communication group will change the system architecture since the user's role is dynamically changing. Therefore, we introduce a communication channel to integrate the information of all agents to determine the communication group. Specifically, the communication group is an LSTM unit, parameterized as θ^g. At the same time, each agent partially observes the environment, and the communication channel can facilitate the sharing of information between the agents to produce more reasonable computation offloading decisions.

When an initiator selects its collaborators, it usually selects users within the range it can observe. Similar users can guarantee lower communication latency, while cooperative decision-making can be more easily accomplished among adjacent agents. When multiple initiators select a user to join their communication group, the user will participate in the communication group of each initiator and act as a bridge for information sharing. Assuming user k is selected by two initiators a and b sequentially. The communication channel integrates their thoughts: $\{\tilde{\mathcal{H}}_a^t, \ldots \tilde{\mathcal{H}}_k^{t\prime}\} = g(\mathcal{H}_a^t, \ldots \mathcal{H}_k^t)$. Then user k communicates with b's group: $\{\tilde{\mathcal{H}}_b^t, \ldots \tilde{\mathcal{H}}_k^{t\prime\prime}\} = g(\mathcal{H}_b^t, \ldots \mathcal{H}_k^{t\prime})$. User k disseminates the thought from one group to other groups, guiding a cooperation strategy among the groups. The pseudo-code of the method is shown in lines 2 to 7 of Algorithm 1.

For each initiator m and its communication group C, the attention mechanism is parameterized by θ^p. We train one critic network θ^c embedded with a attention mechanism for each initiator m. The integrated thought is defined as $\tilde{\mathcal{H}}_m^t$, it is merged with $\mathcal{H}_m^t$ and input to the next attention mechanism policy network. Then the attention mechanism policy network outputs the action intention $\mathbf{ai}_m^t = \mu_2(\mathcal{H}_m^t, \tilde{\mathcal{H}}_m^t; \theta^{\mu_2})$. We calculate the average of the difference in Q-value between independent actions intention $\bar{\mathbf{ai}}_i$ and cooperative actions

$$\Delta Q_m^t = \frac{1}{|C|}(\sum_{i\in C} Q(\mathbf{s}_i, \mathbf{ai}_i|\theta^c) - \sum_{i\in C} Q(\mathbf{s}_i, \bar{\mathbf{ai}}_i|\theta^c)). \tag{7}$$

We store $(\Delta Q_m^t, \mathcal{H}_m^t)$ into a queue $\mathcal{U}$, and perform min-max normalization on ΔQ in $\mathcal{U}$. Then we will get $\Delta\hat{Q} \in [0,1]$, it can be used as the tag of the binary classification. We update θ^p in the loss function as

$$\mathcal{L}(\theta^p) = -\Delta\hat{Q}_m^t log(p(\mathcal{H}_m|\theta^p)) - (1-\Delta\hat{Q}_m^t) log(1-p(\mathcal{H}_m|\theta^p). \tag{8}$$

The pseudo-code of the method is shown in line 8 to line 14 of Algorithm 1.

Algorithm 1. Attention Mechanism Method

Input: Communication group C, a set of UE m^p
Output: initiator m, Communication group C of the initiator

```
1:  for episode $= 1, \ldots$ episode$_{max}$ do
2:    for time slot $= 1, \cdots, \mathcal{T}$ do
3:      Get thought $\mathcal{H}_m^t = \mu_1(\mathbf{s}_t^m; \theta^{\mu_1})$for each initiator $m$
4:      Each agent $m$ decides whether to initiate communication based on $\mathcal{H}_m^t$ every $\mathcal{T}$ time slot
5:      for $i$ in communication group do
6:        $(\tilde{\mathcal{H}}_m^t, \tilde{\mathcal{H}}_1^t, \ldots, \tilde{\mathcal{H}}_i^t) = g(\mathcal{H}_m^t, \mathcal{H}_t^1, \ldots, \mathcal{H}_t^i; \theta^g)$, where agent 1 to $i$ in communication group of initiator $m$
7:      end
8:      Select action intention $\mathbf{ai}_m^t = \mu_2((\mathcal{H}_m^t, \tilde{\mathcal{H}}_m^t; \theta^{\mu_2})$ for each agent $m$ with communication
9:      Select action $\mathbf{ai}_m^t = \mu_2(\mathcal{H}_m^t; \theta^{\mu_2})$ for each agent $m$ without communication
10:     for $m$ =1,... $M$ do
11:       Obtain reward $\mathbf{r}_t^m$ based on $\mathbf{ai}_m^t$, and get new observation $\mathbf{s}_{t+1}$
12:       Get action for $\bar{\mathbf{ai}}_m^t = \mu_2(\mathcal{H}_m^t; \theta^{\mu_2})$ in initiator $m$'s communication group $C$
13:       Compute different of mean $Q$ values with and without communication $\Delta Q_m^t$
14:       Store $(\mathcal{H}_m^t, \Delta Q_m^t)$ in $\mathcal{U}$
15:     end
16:     Store transition $(s, a, r, s')$ in $\mathcal{D}$
17:   end
18:   Compute $\Delta\hat{Q}$
19:   Update the attention parameter $\theta^p$ by (8)
20: end
```

4.2.4 The Training Process

Considering the strategy improvement, we employ the Actor network $\mu(\theta)$ to generate the next state action $\mu(\mathbf{a}_{t+1}|\theta^\mu)$ instead of selecting the action corresponding to the largest Q-value in the action space. Since in such a multi-dimensional action space, the greedy strategy needs to find the global maximum Q-value in every step, it is inadequate to find the global maximum Q-value in such a large action space. Moreover, we employ the Critic network $Q(\theta)$ criticizes the policy according to the estimated Q-value $Q(\mathbf{s}_t, \mathbf{a}_t|\theta^Q) \approx Q(\mathbf{s}_t, \mathbf{a}_t)$ by a method similar to supervised learning, so that the gradient can be written as (process 13 of Fig. 2)

$$\nabla_\theta J(\mu) = E_{\mathbf{s}_t, \mathbf{a}_t \sim \mathcal{D}}[\nabla_\theta \mu(\mathbf{a}_t|\mathbf{s}_t) \nabla Q^\mu(\mathbf{s}_t, \mathbf{a}_t)|_{\mathbf{a}=\mu(\mathbf{s}_t)}], \tag{9}$$

where $\mathcal{D}$ is the experience replay memory containing $\{\mathbf{s}_t, \mathbf{a}_t, \mathbf{r}_t, \mathbf{s}_{t+1}, C\}$, and stores the transitions of all agents. The action value function can be updated as follow:

$$L(\theta) = E_{\mathbf{s}_t, \mathbf{a}_t, \mathbf{r}_t, \mathbf{s}_{t+1}, C}[(Q^\mu(\mathbf{s}_t, \mathbf{a}_t) - y)]^2, \tag{10}$$

where $y = r + \gamma Q^{\mu'}(\mathbf{s}_{t+1}, \mathbf{a}_{t+1})|_{\mathbf{a}_{t+1}=\mu(\mathbf{s}_t)}$.

Algorithm 2. D2D-Assisted Attention Cooperative Multi-Agent Deep Reinforcement Learning

```
Input: $\theta^{\mu_1}$, $\theta^{\mu_2}$, $\theta^{\mu}$, $\theta^{c}$, $\theta^{Q}$, $\theta^{g}$ and $\theta^{p}$ for attention cooperative formulation, $\gamma$ for the target network, $\alpha$ and $\beta$ for PER method.
Output: Computation offloading decision
for Agent $m \in M$ do
  Initialize the target Actor and Critic networks with weights $\theta^{\mu'} \leftarrow \theta^{\mu}$ and $\theta^{Q'} \leftarrow \theta^{Q}$;
  Initialize the queue $\mathcal{U}$ and the replay buffer $\mathcal{D}$
end
for $episode = 1$ to $episode_{max}$ do
  Initialize system environment and obtain state $s_t$
  for $t = 1$ to $\mathcal{T} - 1$ do
    Choose action $a_t \sim \mu_\theta(a_t|s_t)$ for each Agent
  end
  Get initiator and it's communication group $C$ by Attention mechanism method;
  Obtain the next state $s_{t+1}$ and reward $r_t$;
  Store transition $(a_t, s_t, r_t, s_{t+1}, C)$ in $\mathcal{D}$;
  Update $s \leftarrow s_{t+1}$
  If t mod $\mathcal{T}$ =0 then
    Sample a subset $\mathcal{D}'$ from $\mathcal{D}$ by PER
  end
  Set $y = r + \gamma Q^{\mu'}(s_{t+1}, a_{t+1})|_{a'_j=\mu_j(s_j)}$
    For each gradient step do
      Update Actor network by sampling policy gradient using (9)
      Update Critic network by minimizing the loss using (10)
      Update the Critic network using $\theta^Q \leftarrow \theta^Q + \alpha^Q \cdot \nabla_{\theta^\mu} L^Q(\theta^Q)$
      Update the Actor network using $\theta^\mu \leftarrow \theta^\mu + \alpha^\mu \cdot \theta^\mu$
      Update the target networks using (14)
  end
end
```

The Actor network $\mu(\theta)$ can be updated based on the action $\mathbf{a}_t$ evaluated by the Critic network which is generated by the forward transfer of the Actor. Therefore, the iterative formula of the policy gradient $\nabla_{\theta^\mu}\mu$ is used to update the Actor network (processes 12 and 15 of Fig. 2)

$$\nabla_{\theta^\mu}\mu = \nabla_{\mathbf{a}_t} Q(\mathbf{s}_t, \mathbf{a}_t|\theta^Q)_{\mathbf{s}_t=\mathbf{s}_t, \mathbf{a}_t=\mu(\mathbf{s}_t)}. \tag{11}$$

To improve the stability of the learning process, we introduce two target networks $\theta^{\mu'}$ and $\theta^{Q'}$ for Actor and Critic networks to limit the change speed of the target value. Since the Critic network $Q(\mathbf{s}, \mathbf{a}|\theta^Q)$ is also used to calculate the target value $\mathbf{r}_t + \gamma Q(\mathbf{s}_{t+1}, \mu(\mathbf{s}_{t+1})|\theta^\mu))|\theta^Q)$, the update of Q-value is prone to shock. The target network updates the weight according to the way of slowly tracking the online network, $\theta' \leftarrow \tau\theta + (1-\tau)\theta', \tau \ll 1$.

The MEC environment receives and performs the offloading action $\mathbf{a}_t$ returned by the Agent during the learning process (process 16 of Fig. 2). Then, the Agent feeds back a reward $\mathbf{r}_t$ and a next state $\mathbf{s}_{t+1}$. Normally, an experience replay mechanism is used in this process which stores previous experiences $(\mathbf{s}_t, \mathbf{a}_t, \mathbf{r}_t, \mathbf{s}_{t+1}, C)$, then sampling these experiences evenly in small batches $\mathcal{D}'$, and updating the online network of Actor and Critic at each time step (process 7 of Fig. 2). However, most deep learning algorithms assume that these state samples are not independent and identically distributed. In order to improve the efficiency of sample utilization and improve the learning rate, a sample policy called Priority Experience Replay (PER) mechanism is introduced in this method (process 8 of Fig. 2), which can make learning from experience replay more efficient [41]. In this mechanism, the estimate of Temporal-Difference Learning (TD-error) ψ is recorded as Q-value, reflecting the degree to which Agent has learned from current experience. It can be expressed as follow:

$$\psi = \mathbf{r}_t + \gamma[\max_{\mathbf{a}_{t+1}} Q^{\mu'}(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}) - Q^\mu(\mathbf{s}_t, \mathbf{a}_t)]|_{\mathbf{a}_{t+1}=\mu(\mathbf{s}_t)}. \tag{12}$$

The experience of a large TD-error value means that there is still much room for improvement in intensive reading of sample prediction, and Agent can learn a lot from this sample. The TD-error value is very small or even very negative, which means that this behavior is opposite to the correct direction. The application of PER strategy allows Agent to learn from successful experience and prevents Agent from choosing the wrong operation from the bad experience, thereby improving the quality of learning strategy. Therefore, based on TD-error ψ, we define the sampling probability P_m of Agent m as $P_m = \frac{p_m^\alpha}{\sum_j p_j^\alpha}$, where α is the control parameters for sorting quantity for priority, j is the index of the PER batch sample. $p_m = |\psi_m| + \epsilon$, and ϵ is a small positive constant that can prevent the critical case of this transition from reconsidering once an error with zero probability occurs. Because transitions with high P_m values are replayed more often, this practice changes the replay frequency of some samples, thus introducing bias. We can use Importance Sampling (IS) weights to correct this deviation $\eta_m = (\frac{1}{N} \cdot \frac{1}{P_m})^\beta$, where β is to adjust the correction degree. In the process of Q-value learning update, $\eta_m \psi_m$ is used instead of ψ_m. At the same time, for the stability of training, the weight is always normalized as $\frac{1}{max_m \eta_m}$.

Finally, apply the following update to the Actor and Critic networks (process 9 and process 14 of Fig. 2), where α^Q and α^μ are the network learning rates

$$\theta^Q \leftarrow \theta^Q + \alpha^Q \cdot \nabla_{\theta^\mu} L^Q(\theta^Q), \theta^\mu \leftarrow \theta^\mu + \alpha^\mu \cdot \theta^\mu. \tag{13}$$

The target Critic network and target Actor network are updated according to the following way to step-by-step track the online Critic network and Actor network (process 11 of Fig. 2):

$$\theta^{Q\prime} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q\prime}, \theta^{\mu\prime} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu\prime}, \tag{14}$$

where $\tau \ll 1$ is the temperature parameter. The pseudo-code of the method is shown in Algorithm 2.

4.2.5 *Computational Complexity Analysis*

We have defined $\mathcal{T}$ time steps in the reinforcement learning algorithm, i.e., $\mathcal{T} = \{1, 2, \dots t\}$, the computational complexity can be simplify expressed as $O(\mathcal{T}^2)$ [42]. However, it should be noted many parameters i.e., the number of UE, the number of initiators. Therefore, to determine the computational complexity of our algorithm, we analyze the computational complexity of two modules (communication channel and the attention mechanism). Let denote P as the number of UEs in this system, I is the number of the communication group, Q is the dimensionality of the UE's state space, U is the number of experiences sampled in each round of training, and V is the max episode during the training process. For the communication channel, let set the hidden layer dimension to l, according to [43], the complexity of the communication channel is $\mathcal{O}(I \cdot l^2)$. For the attention mechanism, the complexity is $\mathcal{O}(P^2 \cdot Q)$ according to Algorithm 1. Moreover, assume that the computation complexity for the training of one experience is $\mathcal{O}(L)$, where L is the number of multiplication operations in the neural network, the computation complexity of the proposed algorithm is $\mathcal{O}(LI^2P^2QV)$ according to [22].

The actual implementation latency of the system is also crucial in measuring system performance. Based on current hardware conditions, the training process for deep learning induces a significant computational latency, and it is not practical to complete the training on a mobile device [44]. Therefore, we attempt to move the training process to a closer-edge server where computational resources are more abundant. To implement the above strategy, each UE agent uploads historical information collected during execution, i.e., states, actions, and rewards received by the UE agent, to the edge server. The historical information generated by the UE agent in a time slot is only a few kilobytes in size since all of those are digital data, and the communication latency for this part is minimal. Once the training process is complete, the UE downloads the weights of the trained actor network from the edge server. It imports them into its actor network to complete the computational offload decision. Moreover, the weights of the neural networks are digital data, and the size of the actor network for each UE agent is about 400 KB in size, which does not cause much transmission latency. Therefore, transferring the complex training process to the edge server requires only a tiny transmission latency. In addition, compared to traditional strategies, our approach significantly reduces the amount of real-time signaling overhead since it only requires the collaborators within the UE agent's communication group to upload the historical information to the edge server.

5 Simulation Results

5.1 Simulation Settings

In this part, we employ Pytorch 1.3 framework to compare the performance of several methods. Without loss of generality, let's assume a setting with 150 UEs and 8 ENs in this MEC computation offloading environment. There are 45 collaborators for each initiator in its communication group. Throughout the experiments, we suppose a scenario where UEs are randomly distributed within an area of $350m \times 350m$. Similar to [26], we consider that the different perform distinct computation capabilities of UE g_m, uniformly distributed between 0.5 and 3.5 GHz, the computation capabilities of EN f_n are distributed between 31.5 and 51.5 GHz. The channel bandwidths between the UEs and ENs range from [4, 20] MHz, and the bandwidth among UEs ranges from [100, 200] kHz according to [18], [22]. The Key evaluation parameters are listed in Table 2.

TABLE 2
Parameters Setting

Parameter	Value
M^p	20
N	8
Δ	0.01s
g_m, q_i	1.5 $\sim$ 3.5 GHz[26]
f_n	31.5 $\sim$ 51.5GHz[26]
$b^{e,t}_{m,n}$	4$\sim$20 MHz[18]
$b^{u,t}_{m,i}$	100$\sim$200 KHz[18]
c^t_m	100$\sim$1000 KB[18]
PER batch	256
Episode	3000

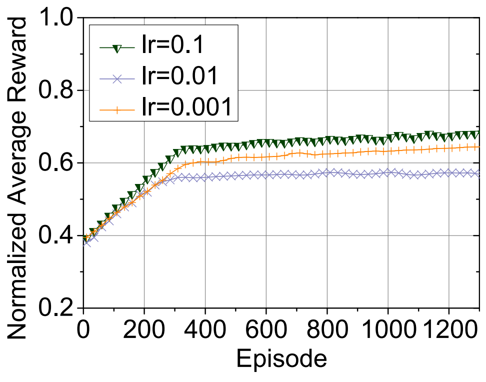
Without loss of generality, for the task execution, the task tolerance latency follows the uniform QoS between [5, 30] seconds, and the waiting time at ENs queue ranges from [0, 40] seconds. While the task data sizes follow the uniform distribution on [100, 1000] KB, the requested CPU z^t_m ranges from [10, 50] G cycles according to [45]. Moreover, the pass-loss constant L is -2, the channel gain between UE and EN $h^t_{m,n}$ and among UEs $h^t_{m,i}$ range from [-20, -5] dB and [-5, -10] dB [39], respectively. For the design of the ACDRL, we set the size of experience replay memory $\mathcal{D}$ as 1,024 [22]. There are three different sizes for batch $\mathcal{D}'$, which are 128, 256, and 512. For the parameters of ACDRL, the target network parameter is 0.8, α and β for the PER method as 0.9 and 0.9 [26], respectively. For the parameters of the target network, we set the α^Q, α^μ and τ for PER method as 0.9, 0.9, and 0.001 respectively, which have been verified to be suitable for DRL [18], [22], [46].

5.2 Convergence Analysis

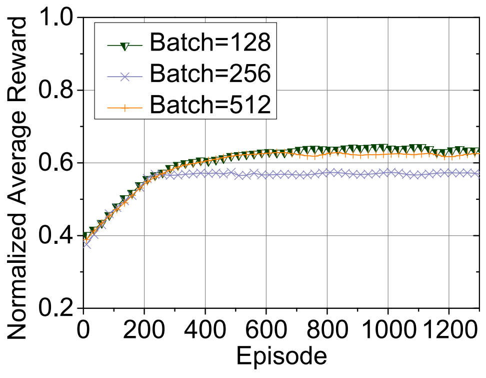
To demonstrate the validity of the proposed method in terms of convergence, we perform the simulation experiment with $M^p = 20$, $N = 8$, a random tolerance latency $\delta^t_m \in [5, 20]$ seconds for each task.

5.2.1 *Episode versus Average Episode Reward*

We evaluate the convergence of the proposed algorithm ACDRL under different experimental settings. The simulation results are shown in Fig. 3. In the subfigures, the x-axis shows the episode, and the y-axis shows the normalized average reward in each episode. We plot the performance of the comparison algorithms and ACDRL under different parameter settings.



(a)



(b)

Fig. 3. Performance under different parameter setting: (a) learning rate; (b) batch size.

Fig. 3a shows the convergence of our algorithm under different values of learning rate (“lr” in this figure), where the learning rate is the step that moves toward the minimum value of the loss function in each iteration. It can be shown in this figure, when lr = 0.01, the convergence rate is relatively fast, and the average reward converges to the smallest value. When the learning rate is small, the convergence rate slows down and the average reward increases. When the learning rate is big, the average reward is higher. Fig. 3b shows the convergence of our proposed algorithm at different batch sizes. The batch refers to the amount of experience each selection step under the PER strategy. It can be shown in this figure, when batch = 256, the convergence is relatively fast and the average reward is smaller than other algorithms. The convergence rate will slow down when the size of batch increases or decreases, and the average reward of convergence will increase.

5.2.2 *Performance of Execution and Convergence Time*

Fig. 4 shows the effect of the different numbers of UEs on execution delay and convergence time. As the number of UEs increases, the execution time and convergence time increase, which is consistent with the previous theoretical analysis. The line graph shows that when the number of UE increases, the execution time of the proposed algorithm becomes longer. This is because more UE can generate more tasks, and more tasks consume more time for scheduling. The bar graph shows that its convergence time is acceptable

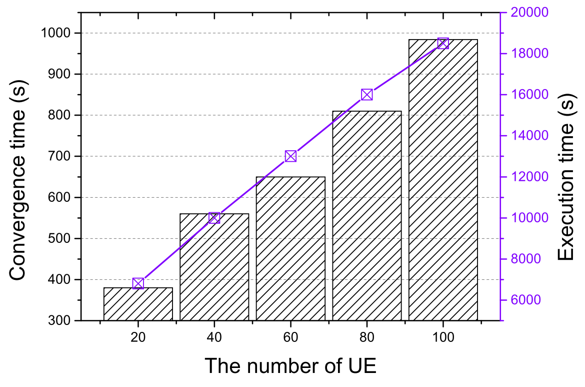


Fig. 4. Convergence time and execution time as the number of UEs varies.

compared with its total execution time. Similarly, the convergence time of the designed algorithm grew gradually as the number of UEs increase. This is because more UEs can generate more tasks, and the algorithm execution time becomes longer, correspondingly.

5.2.3 *Effect of Communication Group Size*

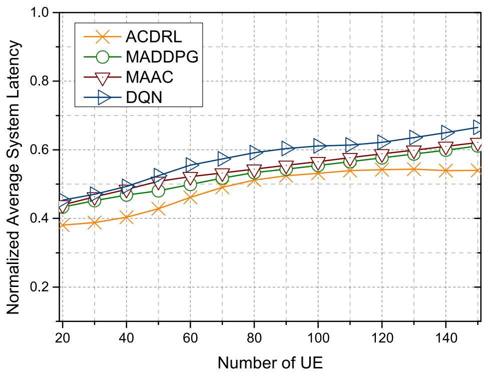
A larger communication group will cause invalid training data to increase system latency (i.e., all UE in this system are collaborators), and a small communication group will cause the most appropriate UE for offloading to be lost. Therefore, we explore the size of the communication group I effects on the average system reward. We compare three sizes of the communication group by 10%, 30%, and 50% of the number of UE M^p, and change the number of UE M^p from 20 to 100 for comparison. As shown in Table 3, when the size of the communication group is 10% of UEs M^p, the average system reward increases as the number of UE increases, when the size of the communication group is increased to 30% of M^p, the average system reward decreases slightly. However, when the number of UE increases to 50% of M^p, the average system reward increases again. It proves our conjecture that the average system reward will not decrease with the size of the communication group increase, and choosing an appropriate communication group can effectively reduce system reward.

5.3 Method Comparison

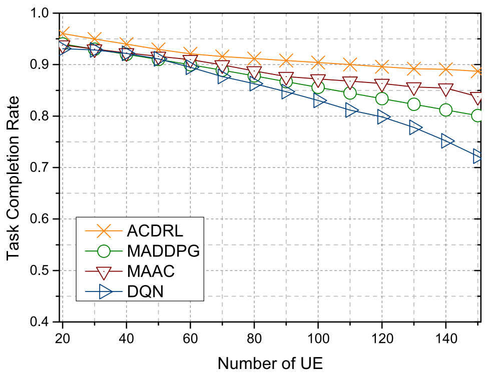
In this section, we evaluate the presented intelligent offloading strategy by comparing it with several benchmarks, including MADDPG [39], MAAC-based algorithm [18], and DQN-based algorithm [46]. The MADDPG, MAAC, and DQN are widely used in MDP problems. The DQN-based

TABLE 3
Normalized Average System Reward

size \ M^p	20	40	60	80	100
10%	0.3933	**0.3998**	0.4619	0.5167	0.5494
30%	**0.3806**	0.4034	**0.4614**	**0.5114**	**0.5312**
50%	0.3812	0.4335	0.4767	0.5237	0.5563

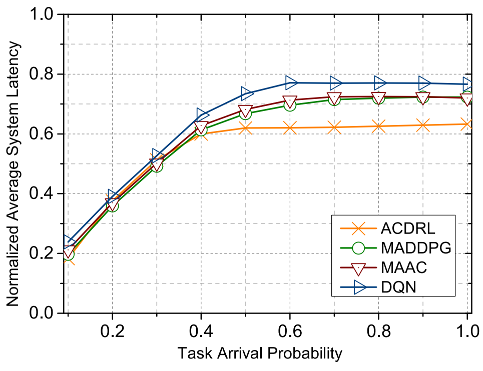


(a)

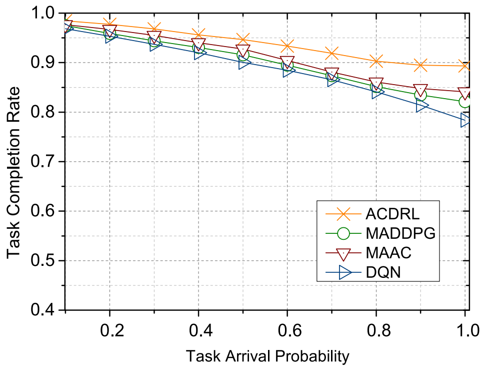


(b)

Fig. 5. Performance evaluation under different number of UE with (a) average system latency; (b) task completed rate.



(a)



(b)

Fig. 6. Performance evaluation under different task arrival probability with: (a) Average system latency; (b) Task completed rate.

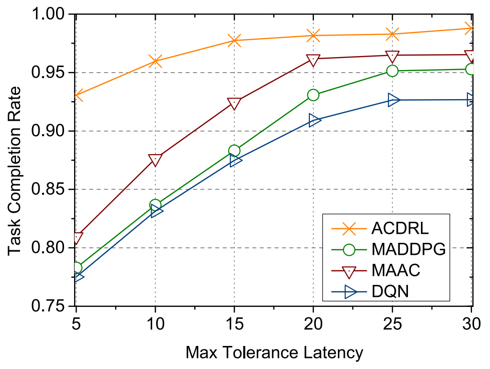
algorithm can be regarded as a single agent to make the computation offloading dependently. For the MADRL, we adopt the state-of-art MADRL framework MADDPG approach. Moreover, the MAAC-based algorithm is also a multi-agent reinforcement learning based on actor-critic like our algorithm.

We look at how the number of UEs affects the average system latency of these computation offloading algorithms. In Fig. 5a, as the amount of UEs grows, the average system latency of each algorithm increases. More UEs compete for transmission and computing resources at edge nodes when the number of UE increases. When the number of UE increases to 100, our algorithm maintains a ratio of average system latency around 0.53 less than other algorithms. This is because our algorithm ignores meaningless information in the system. It reduces the dimension of the state space and improves computational efficiency. Moreover, our algorithm can reduce communication latency effectively. When the number of UE increases to 150, it achieves an average system latency of 11% lower than those of other algorithms.

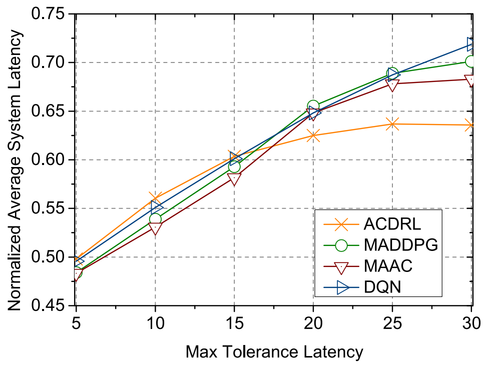
Next, as shown in Fig. 5b, the proposed algorithm always receives a higher task completion rate than the other algorithms, especially in the case of a large number of UEs. When the number of UE is 150, the proposed algorithm increases the task completed rate by 17% compared with the benchmark methods. This is because the computation offloading method based on the ACDRL method integrates all agent information in the communication group, improving cooperation's effectiveness and rationality. Therefore, our scheme improves the computation task by choosing the most effective offloading decision and completing it within the tolerance latency.

In Fig. 6a, as the task arrival probability increase, the proposed algorithm always maintains a lower average reward when compared with the benchmark algorithms. When the task probability is small, most methods can achieve an average reward of around 0.2. When the task arrival probability increases 1.0, the average reward of our algorithm increases by 40.1%, while those of the benchmark methods increase by at least 52%. This implies that as loads of the system increase, especially at peak-time traffic, the average reward of the proposed algorithm increases less than those of the benchmark algorithms.

As shown in Fig. 6b, the proposed algorithm achieves a higher task completion rate than the benchmark algorithms, especially when a large task arrival probability. This is because the proposed algorithm can effectively use the idle computation resources at the network by D2D links. When the task arrival probability increases from 0.1 to 1.0, the task completed rate of the proposed algorithm remains higher than 0.89, while those of the benchmark methods decrease from 0.97 to 0.76. This is because when the task arrival probability increases, the limited computation resources cannot meet the requirements of all tasks. Therefore, some tasks may be dropped since they are incomplete within their max tolerance latency. This implies that as the load of the system increases, our algorithm may have a more significant task completed rate than the other algorithms, as it has fewer tasks dropped.



(a)



(b)

Fig. 7. Performance evaluation under different max tolerance latency (in seconds) with: (a) task completed rate; (b) average system latency.

We evaluate the performance of the system under different max tolerance latency and algorithm settings in Fig. 7a. It shows that as the max tolerance latency increases from 5 to 30 seconds, the task completed rate of the four methods increases significantly. As expected, it can be seen that our proposed algorithm can maintain a higher task completion rate than other algorithms. Specifically, the task competition rate of ACDRL is much higher than other algorithms, mainly while the max tolerance latency is small. This verifies that our algorithms are more excellent in time-sensitive tasks than other algorithms.

Similarly, Fig. 7b shows that as the max tolerance latency increases from 5 to 30 seconds, the average system latency increases significantly with four methods. This is because most tasks cannot be completed when the max tolerance latency is small. As the max tolerance latency is prolonged to 30 seconds, the task has enough time to process and transmit, and the average system latency increases for this reason.

6 Conclusion

In this article, we studied the computation offloading problem in a D2D-assisted communication framework and proposed an attention communication multi-agent reinforcement learning model to optimize offloading decisions by minimizing the average system latency in dynamic users MEC environment. In this system, the attention mechanism lets the computing resource tilt to active users to adapt to a dimensional explosion and reduce the communication overhead in a dynamic MEC environment. Moreover, the communication channel integrates all agents information in the communication group, thereby facilitating collaborative decision-making, which improves the effectiveness and rationality of cooperation. Numerical results show that the proposed ACDRL method is effective and superior to the baseline algorithms in moving edge computation offloading.

In our future work, we will design architecture to introduce an incentive mechanism to encourage mobile devices to deal with the computation offloading problem. At the same time, we will consider UEs' computing service cost (such as price) to adapt the actual MEC system.

References

[1] B. Ji et al., "A survey of computational intelligence for 6G: Key technologies, applications and trends," *IEEE Trans. Ind. Informat.*, vol. 17, no. 10, pp. 7145–7154, Oct. 2021.
[2] S.-W. Ko, K. Han, and K. Huang, "Wireless networks for mobile edge computing: Spatial modeling and latency analysis," *IEEE Trans. Wireless Commun.*, vol. 17, no. 8, pp. 5225–5240, Aug. 2018.
[3] C. Jiang, X. Cheng, H. Gao, X. Zhou, and J. Wan, "Toward computation offloading in edge computing: A survey," *IEEE Access*, vol. 7, pp. 131 543–131 558, 2019.
[4] J. Zheng, Y. Cai, Y. Wu, and X. Shen, "Dynamic computation offloading for mobile cloud computing: A stochastic game-theoretic approach," *IEEE Trans. Mobile Comput.*, vol. 18, no. 4, pp. 771–786, Apr. 2019.
[5] B. Yang, X. Cao, J. Bassey, X. Li, and L. Qian, "Computing offloading in multi-access edge computing a multi-task learning approach," *IEEE Trans. Mobile Comput.*, vol. 20, no. 9, pp. 2745–2762, Sep. 2021.
[6] F. Saeik et al., "Task offloading in edge and cloud computing: A survey on mathematical, artificial intelligence and control theory solutions," *Comput. Netw.*, vol. 195, 2021, Art. no. 108177.
[7] X. Wang, R. Li, C. Wang, X. Li, T. Taleb, and V. C. M. Leung, "Attention-weighted federated deep reinforcement learning for device-to-device assisted heterogeneous collaborative edge caching," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 1, pp. 154–169, Jan. 2021.
[8] U. Saleem, Y. Liu, S. Jangsher, X. Tao, and Y. Li, "Latency minimal for D2D-enabl partial computing offloading in mobile edge computing," *IEEE Trans. Veh. Technol.*, vol. 69, no. 4, pp. 4472–4486, Apr. 2020.
[9] Y. Yang, C. Long, J. Wu, S. Peng, and B. Li, "D2D-enabled mobile-edge computing offloading for multiuser IoT network," *IEEE Internet Things J.*, vol. 8, no. 16, pp. 12 490–12 504, Aug. 2021.
[10] Y. Pan, C. Pan, Z. Yang, M. Chen, and J. Wang, "A caching strategy towards maximumal D2D assisted offload gain," *IEEE Trans. Mobile Comput.*, vol. 19, no. 11, pp. 2489–2504, Nov. 2020.
[11] M. Hamdi, A. B. Hamed, D. Yuan, and M. Zaied, "Energy-efficient joint task assignment and power control in energy harvesting D2D offload communications," *IEEE Internet Things J.*, vol. 9, no. 8, pp. 6018–6031, Apr. 2022.
[12] M. Sun, X. Xu, X. Tao, and P. Zhang, "Large-scale user-assisted multi-task online offloading for latency reduction in D2D-enabled heterogeneous networks," *IEEE Trans. Netw. Sci. Eng.*, vol. 7, no. 4, pp. 2456–2467, Fourth Quarter 2020.
[13] L. Ryan, W. Yi, T. Aviv, H. Jean, A. Pieter, and M. Igor, "Multiagent actor-critic for mixed cooperative-competitive environments," in *Proc. 31st Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 6382–6393.
[14] A. Khan, C. Zhang, D. D. Lee, V. Kumar, and A. Ribeiro, "Scalable centralized deep multi-agent reinforcement learning via policy gradients," 2018, *arXiv:1805.08776*.
[15] H. Peng and X. Shen, "Multi-agent reinforcement learning based resource management in MEC- and UAV-assisted vehicular networks," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 1, pp. 131–141, Jan. 2021.

[16] Z. Chen, L. Zhang, Y. Pei, C. Jiang, and L. Yin, "NOMA-based multiuser mobile edge computation offloading via cooperative multiagent deep reinforcement learning," *IEEE Trans. Cogn. Commun. Netw.*, vol. 8, no. 1, pp. 350–364, Mar. 2022.

[17] A. Sacco, F. Esposito, G. Marchetto, and P. Montuschi, "Sustainable task offloading in UAV networks via multi-agent reinforcement learning," *IEEE Trans. Veh. Technol.*, vol. 70, no. 5, pp. 5003–5015, May 2021.

[18] Z. Li and C. Guo, "Multi-agent deep reinforcement learning based spectrum allocation for D2D underlay communications," *IEEE Trans. Veh. Technol.*, vol. 69, no. 2, pp. 1828–1840, Feb. 2020.

[19] B. Huang, X. Liu, S. Wang, L. Pan, and V. Chang, "Multi-agent reinforcement learning for cost-aware collaborative task execution in energy-harvesting D2D networks," *Comput. Netw.*, vol. 195, 2021, Art. no. 108176.

[20] D. Shi, L. Li, T. Ohtsuki, M. Pan, Z. Han, and V. Poor, "Make smart decisions faster: Deciding D2D resource allocation via stackelberg game guided multi-agent deep reinforcement learning," *IEEE Trans. Mobile Comput.*, vol. 21, no. 12, pp. 4426–4438, Dec. 2022.

[21] Q. He et al., "Reinforcement-learning-based competitive opinion maximization approach in signed social networks," *IEEE Trans. Comput. Social Syst.*, vol. 9, no. 5, pp. 1505–1514, Oct. 2022.

[22] M. Tang and V. W. Wong, "Deep reinforcement learning for task offloading in mobile edge computing systems," *IEEE Trans. Mobile Comput.*, vol. 21, no. 6, pp. 1985–1997, Jun. 2022.

[23] X.-Q. Pham, T.-D. Nguyen, V. Nguyen, and E.-N. Huh, "Joint service caching and task offloading in multi-access edge computing: A QoE-based utility optimization approach," *IEEE Commun. Lett.*, vol. 25, no. 3, pp. 965–969, Mar. 2021.

[24] G. Li, M. Chen, X. Wei, T. Qi, and W. Zhuang, "Computation offloading with reinforcement learning in D2D-MEC network," in *Proc. Int. Wireless Commun. Mobile Comput.*, 2020, pp. 69–74.

[25] H. Zhou, T. Wu, H. Zhang, and J. Wu, "Incentive-driven deep reinforcement learning for content caching and D2D offloading," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 8, pp. 2445–2460, Aug. 2021.

[26] J. Tang, H. Tang, N. Zhao, K. Cumanan, S. Zhang, and Y. Zhou, "A reinforcement learning approach for D2D-assisted cache-enabled hetnets," in *Proc. IEEE Glob. Commun. Conf.*, 2019, pp. 1–6.

[27] Y. Lan, X. Wang, D. Wang, Z. Liu, and Y. Zhang, "Task caching, offloading, and resource allocation in D2D-aided fog computing networks," *IEEE Access*, vol. 7, pp. 104 876–104 891, 2019.

[28] X. Chen, L. Pu, L. Gao, W. Wu, and D. Wu, "Exploiting massive D2D collaboration for energy-efficient mobile edge computing," *IEEE Wireless Commun.*, vol. 24, no. 4, pp. 64–71, Aug. 2017.

[29] Y. Yang, C. Long, J. Wu, S. Peng, and B. Li, "D2D-enabled mobile-edge computation offloading for multi-user IoT network," *IEEE Internet Things J.*, vol. 8, no. 16, pp. 12490–12504, Aug. 2021.

[30] Q. Lin, F. Wang, and J. Xu, "Optimal task offloading scheduling for energy efficient D2D cooperative computing," *IEEE Commun. Lett.*, vol. 23, no. 10, pp. 1816–1820, Oct. 2019.

[31] X. Chen, C. Wu, Z. Liu, N. Zhang, and Y. Ji, "Computation offloading in beyond 5G networks: A distributed learning framework and applications," *IEEE Wireless Commun.*, vol. 28, no. 2, pp. 56–62, Apr. 2021.

[32] X. Wang, C. Wang, X. Li, V. C. M. Leung, and T. Taleb, "Federated deep reinforcement learning for Internet of Things with decentralized cooperative edge caching," *IEEE Internet Things J.*, vol. 7, no. 10, pp. 9441–9455, Oct. 2020.

[33] D. Shi, H. Gao, L. Wang, M. Pan, Z. Han, and H. V. Poor, "Mean field game guided deep reinforcement learning for task placement in cooperative multiaccess edge computing," *IEEE Internet Things J.*, vol. 7, no. 10, pp. 9330–9340, Oct. 2020.

[34] L. Pu, X. Chen, J. Xu, and X. Fu, "D2D fogging: An energy-efficient and incentive-aware task offloading framework via network-assisted D2D collaboration," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 12, pp. 3887–3901, Dec. 2016.

[35] X. Qiu, W. Zhang, W. Chen, and Z. Zheng, "Distributed and collective deep reinforcement learning for computation offloading: A practical perspective," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 5, pp. 1085–1101, May 2021.

[36] X. Cao et al., "Massive access of static and mobile users via reconfigurable intelligent surfaces: Protocol design and performance analysis," *IEEE J. Sel. Areas Commun.*, vol. 40, no. 4, pp. 1253–1269, Apr. 2022.

[37] Q. Qi et al., "Knowledge-driven service offloading decision for vehicular edge computing: A deep reinforcement learning approach," *IEEE Trans. Veh. Technol.*, vol. 68, no. 5, pp. 4192–4203, May 2019.

[38] G. Gao, M. Xiao, J. Wu, H. Huang, S. Wang, and G. Chen, "Auction-based VM allocation for deadline-sensitive tasks in distributed edge cloud," *IEEE Trans. Serv. Comput.*, vol. 14, no. 6, pp. 1702–1716, Nov./Dec. 2021.

[39] W. Hou, H. Wen, H. Song, W. Lei, and W. Zhang, "Multiagent deep reinforcement learning for task offloading and resource allocation in cybertwin-based networks," *IEEE Internet Things J.*, vol. 8, no. 22, pp. 16 256–16 268, Nov. 2021.

[40] P. Rodríguez, D. Velazquez, G. Cucurull, J. M. Gonfaus, F. X. Roca, and J. Gonzàlez, "Pay attention to the activations: A modular attention mechanism for fine-grained image recognition," *IEEE Trans. Multimedia*, vol. 22, no. 2, pp. 502–514, Feb. 2020.

[41] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," in *Proc. 5th Int. Conf. Learn. Representations*, 2016.

[42] A. Omidkar, A. Khalili, H. H. Nguyen, and H. Shafiei, "Reinforcement-learning-based resource allocation for energy-harvesting-aided D2D communications in IoT networks," *IEEE Internet Things J.*, vol. 9, no. 17, pp. 16 521–16 531, Sep. 2022.

[43] A. Sherstinsky, "Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network," *Physica D: Nonlinear Phenomena*, vol. 404, 2020, Art. no. 132306.

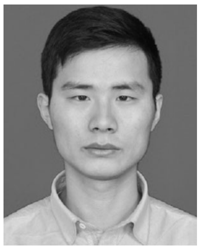
[44] Z. Gao, L. Yang, and Y. Dai, "Large-scale computation offloading using a multi-agent reinforcement learning in heterogeneous multi-access edge computing," *IEEE Trans. Mobile Comput.*, to be published, doi: 10.1109/TMC.2022.3141080.

[45] S. Jošilo and G. Dán, "Wireless and computing resource allocation for selfish computation offloading in edge computing," in *Proc. IEEE Conf. Comput. Commun.*, 2019, pp. 2467–2475.

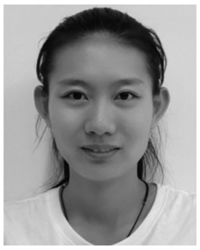
[46] H. Ye, G. Y. Li, and B.-H. F. Juang, "Deep reinforcement learning based resource allocation for V2V communications," *IEEE Trans. Veh. Technol.*, vol. 68, no. 4, pp. 3163–3173, Apr. 2019.

Kexin Li received the BS degree in software engineering from the Harbin University of Science and Technology, Harbin, China, in 2015, and the MS degree in computer technology from Northeastern University, Shenyang, China, in 2018, where she is currently working toward the PhD degree in computer application technology. Her research interests include software-defined networking, edge computing, and machine learning.

Xingwei Wang received the BS, MS, and PhD degrees in computer science from Northeastern University, Shenyang, China, in 1989, 1992, and 1998, respectively. He is currently a professor with the College of Computer Science and Engineering, Northeastern University. He has published more than 100 journal articles, books and book chapters, and refereed conference papers. His research interests include cloud computing and future Internet. He has received several best paper awards.



Qiang He received the PhD degree in computer application technology from Northeastern University, Shenyang, China, in 2020. He also worked with the School of Computer Science and Technology, Nanyang Technical University, Singapore, as a visiting PhD researcher from 2018 to 2019. He has published more than ten journal articles and conference papers. His research interests include social network analytic, machine learning, data mining, and software-defined networking.



Mingzhou Yang received the BSc degree in computer science and technology from the Shenyang University of Technology, Shenyang, China, in 2015, and the MSc degree in computer software and theory from Northeastern University, Shenyang, China, in 2017. She is currently working toward the PhD degree in computer application technology with Northeastern University, Shenyang, China. Her research interests include social network analysis and computational intelligence.

Min Huang (Member, IEEE) received the BS degree in automatic instrument, the MS degree in systems engineering, and the PhD degree in control theory from Northeastern University, Shenyang, China, in 1990, 1993, and 1999, respectively. She is currently a professor with the College of Information Science and Engineering, Northeastern University. She has published more than 100 journal articles, books, and refereed conference papers. Her research interests include modeling and optimization for logistics and supply chain system.

Schahram Dustdar (Fellow, IEEE) is a full professor of computer science (informatics) with a focus on Internet Technologies heading the Distributed Systems Group with the TU Wien. He is chairman of the Informatics Section of the Academia Europaea (since December 9, 2016). He is a member of the IEEE Conference Activities Committee (CAC) (since 2016), the Section Committee of Informatics of the Academia Europaea (since 2015), a member of the Academia Europaea: The Academy of Europe, Informatics Section (since 2013). He is the recipient of the ACM Distinguished Scientist Award (2009) and the IBM Faculty Award (2012). He is an associate editor of *IEEE Transactions on Services Computing*, *ACM Transactions on the Web*, and *ACM Transactions on Internet Technology*, and on the editorial board of *IEEE Internet Computing*. He is the editor-in-chief of *Computing* (an SCI-ranked journal of Springer).