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 paper 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 paper, 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. Secondly, 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, task computation offloading, reinforcement learning, 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 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]–[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]–[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]–[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 Attention 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. Firstly, 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. Secondly, 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 paper 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 prob-
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 multi-helper 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 Sec 3.1, and Sec 3.2 introduces the system model, including local computing, edge computing, and D2D-assisted computing architecture. Sec 3.3 models the problem formulation. 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.
The computation capability and the bandwidth resources of each ES has more excellent computation capacity than UEs. A server (ES); BS is mainly used for communication while each ES deploys a Base station (BS) and an edge server (ES), where each EN deploys a BS and an edge server. The network architecture of D2D-assisted consists of two entities: The Edge Nodes (ENs) and UEs.

1. In this system, the operational timeline in our system is discretized into time slot $T = \{1, 2, ... \}$, where each time slot has a duration of $\Delta$ seconds. The duration matches the timescale at which task offloading could be updated [34]. The network architecture of D2D-assisted consists two entities: The Edge Nodes (ENs) and UEs.

EN: We consider a fundamental model consisting of a set $N = \{1, 2, ... n\}$ 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 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 $M^p = \{1, 2, ... m^p\}$ 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 $M = \{1, 2, ... m\}$, and a set of collaborators $I = \{1, 2, ... i\}$, 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 $\theta_t$ 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 $c_m^t$, $z_m^t$, $\delta_m^t$, 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 $f$ collaborators to form a communication group $C$. Secondly, 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 Sec 4.2 and Sec 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 paper, we assume that each computational 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 $c_m^t$, and $z_m^t$. 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 $\rho_m$ 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
Fig. 1. System model of user cooperation MEC

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 \( \sigma_{m,n}^{i} \) 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 \( c_{m} \), the distance \( d_{m,n} \) and the transmission rate \( r_{m,n} \). 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 \( b_{m,n}^{c} \) 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 \( b_{m,n}^{c} = B m \sum_{m, n} p_{m,n} \), \( m \in M, n \in N \), where \( p_{m,n} \) is the bandwidth allocation factors based on realistic conditions. When \( p_{m,n} = 1 \), the communication resources of EN \( n \) are equally allocated to initiators. In terms of the Shannon Theorem [37], the transmission rate \( r_{m,n}^{i} \) from initiator \( m \) to EN \( n \) can be calculated by:

\[
r_{m,n}^{i} = b_{m,n}^{c} \cdot \log_{2} \left( 1 + \frac{g_{m,n}^{i} h_{m,n}^{i}}{L * d_{m,n}^{c} + \zeta^2} \right) \tag{1}
\]

where \( g_{m,n}^{i} \) is the transmitting power, \( h_{m,n}^{i} \) is the antenna gain at EN \( n \), \( L \) is the path loss at a unit distance and \( d_{m,n}^{c} \) is the distance between initiator \( m \) and EN \( n \). \( \zeta^2 \) 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 \( x_{m}^{i} \) can be expressed as \( D_{m,n}^{tr}(t) = c_{m}^{i} / r_{m,n}^{i} \). If the computational task \( x_{m}^{i} \) is selected as offloading to EN \( n \), the execution delay \( D_{m,n}^{ex}(t) \) can be given as \( D_{m,n}^{ex}(t) = s_{m}^{i} / f_{c} \), where \( f_{c} \) 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, \( D_{m,n}^{ex}(t) < D_{m,n}^{tr}(t) \) 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 \( D_{m,n}^{t}(t) = D_{m,n}^{tr}(t) + D_{m,n}^{ex}(t) + \sigma_{m,n}^{i} \).

Furthermore, we can obtain the transmission energy consumption for initiator offloading the task \( x_{m}^{i} \) to EN \( n \) is \( e_{m,n}^{tr}(t) = c_{m}^{i} / r_{m,n}^{i} \). And obtain the execution consumption for edge computing processing at EN \( n \) as

\[
e_{m,n}^{ex}(t) = \rho_{m} s_{m}^{i} / f_{c} \tag{2}
\]

where \( \rho_{m} \) is the power coefficient of energy consumed per CPU cycle at EN \( n \). Therefore, we can obtain the total energy consumption \( E_{m,n}^{c}(t) \) for edge computing offloading the task \( x_{m}^{i} \) to EN \( n \) as

\[
E_{m,n}^{c}(t) = e_{m,n}^{tr}(t) + e_{m,n}^{ex}(t) \tag{3}
\]

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 SNR\(_{m,i}^{c}\) 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 SNR\(_{m,i}^{c}\) can be formulated as follow:

\[
\text{SNR}_{m,i}^{c} = \frac{(1 + \frac{g_{m,i}^{c} h_{m,i}^{c}}{|\sigma_{i}^{2}|}) \rho_{m}^{2}}{\sigma_{i}^{2}}, i \in M, m \in M \tag{2}
\]

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

\[
r_{m,i}^{c} = b_{m,n}^{c} \cdot \log_{2} \left( 1 + \text{SNR}_{m,i}^{c} \right) \tag{3}
\]

Therefore, the D2D-assisted computation offloading transmission delay \( D_{m,i}^{tr}(t) \) for task \( x_{m,i}^{c} \) of initiator \( m \) to fetch the content through D2D communication can be calculated as \( D_{m,i}^{tr}(t) = c_{m,i}^{c} / r_{m,i}^{c} \), where \( c_{m,i}^{c} \) is the size of content \( x_{m,i}^{c} \). If the computation task will be offloaded to collaborator \( i \), the execution delay can be given as \( D_{m,i}^{ex}(t) = s_{m,i}^{c} / f_{c} \), where \( f_{c} \) 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 \( D_{m,i}^{t}(t) = D_{m,i}^{tr}(t) + D_{m,i}^{ex}(t) \). Unlike the Edge computing 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 \( \lambda_{m,i}^{c} \) for each collaborator to ensure their tasks can be executed timely, which means that the computation tasks need to be completed within \( \lambda_{m,i}^{c} \) 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 \( x_{m}^{i} \) to collaborator \( i \) is \( E_{m,i}^{c}(t) = e_{m,i}^{tr}(t) + e_{m,i}^{ex}(t) \), where \( e_{m,i}^{tr}(t) = c_{m,i}^{c} / r_{m,i}^{c} \) and \( e_{m,i}^{ex}(t) = \rho_{i} s_{m,i}^{c} \), and \( \rho_{i} \) 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 \( (O_{m,n}^t, \gamma_{m,n}^t, Z_{m,i}^t) \) as the offloading status of task \( x_{m}^t \), where \( O_{m,n}^t, \gamma_{m,n}^t, Z_{m,i}^t \in \{0, 1\}, O_{m}^t + \gamma_{m,n}^t + Z_{m,i}^t = 1 \). \( O_{m}^t = 1 \) means the task will be executed locally, \( O_{m}^t = 0 \) means the task will be executed at EN or collaborator. And \( \gamma^t \) indicates whether EN \( n \) is selected to perform the computation offloaded task, \( Z_{m,i}^t \) indicates whether collaborator \( i \) is selected to perform the task; otherwise, \( Z_{m,i}^t = 0 \). Therefore, for a task generated by initiator \( m \), the total task delay can be denoted as:

\[
D_{m}^t = O_{m}^t \cdot D_{m}^t(t) + \gamma_{m,n}^t \cdot D_{m,n}^t(t) + Z_{m,i}^t \cdot D_{m,i}^t(t) \tag{4}
\]

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

Minimize \[
\sum_{m=1}^{M} \sum_{t=1}^{T} D_{m}^t, \forall m \in M, \forall t \in T \tag{5a}
\]

s.t. \( C1: O_{m}^t, \gamma_{m,n}^t, Z_{m,i}^t \in \{0, 1\}, \forall m \in M, \forall t \in T \tag{5b} \)

\( C2: D_{m}^t(t) < \lambda_{i}^t, \forall m \in M, \forall i \in I \tag{5c} \)

\( C3: D_{m}^t(t) < \sigma_{m}^t, \forall m \in M, \forall n \in N \tag{5d} \)

\( C4: D_{m}^t(t), D_{m,n}^t(t) < \delta_{m}, \forall m \in M \tag{5e} \)

\( C5: E_{m}^t(t), E_{m,n}^t(t), E_{m,i}^t(t) < E_{\text{budget}}, \forall m \in M \tag{5f} \)

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 \( x_{1}, x_{2}, ..., x_{M} \) with the energy budget \( E_{\text{budget}} \), 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:

Minimize \[
\sum_{m=1}^{M} E_{m}^t \tag{6a}
\]

s.t. \( \sum_{m=1}^{M} E_{m}^t \leq E_{\text{budget}} \tag{6b} \)

We introduce the trivial 0-1 knapsack problem [38]: “maximize: \( \sum_{i=1}^{n} w_i \cdot x_i \) subject to: \( \sum_{i=1}^{n} v_i \cdot x_i \leq C \), \( x_i \in \{0, 1\} \).” Here, \( w_i \) and \( v_i \) denote the weight and volume of the \( i \)th item, and \( C \) means the capacity of the knapsack. By mapping \( w_i \) and \( v_i \) and \( C \) in the trivial 0-1 knapsack problem to \( D_{m}, E_{m}, \) and \( E_{\text{budget}} \) 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 cannot 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 Sec 4.1, and the details of the training and the whole flow in the processed ACDRL framework are introduced in Sec 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 \( \pi \) for each agent and minimizing the long-term reward (system latency) \( r_t \) which interacts with the MEC environment. The policy \( \pi \) is a mapping from its states \( s_t \) to its action \( a_t \). 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 slot, 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 \( s_t = \{x_t, \epsilon_t, \mathcal{F}, r_{D2D}^t, \gamma_t, \lambda_t, \sigma_t, \mathcal{Q}_t, c_t, \mathcal{C}_t\} \), where \( x_t = \{x_{1}, x_{2}, ..., x_{m}\} \) denotes the state of each UE at time slot \( t \), \( \epsilon_t = \{c_1, c_2, ..., c_m\} \) denotes the state of collaborators of communication group, where \( c_i = \{\lambda_i, \sigma_i\} \), \( \lambda_i = \{\lambda_{1}, \lambda_{2}, ..., \lambda_{I}\} \) denotes the maximum amount of time that can be occupied of collaborator \( i \) at time \( t \), and \( \mathcal{Q}_t = \{q_1, q_2, ..., q_{q}\} \) denotes the computation capability of collaborators. \( \mathcal{C}_t = \{\mathcal{F}, c_t\} \) denotes the states of ENs, where \( \mathcal{F} = \{f_1, f_2, ..., f_n\} \) denotes the computation capability of ENs, and \( c_t = \{c_{1}, c_{2}, ..., c_{m}\} \) denotes the waiting time of EN \( n \) at time slot \( t \). \( r_{D2D}^t = \{r_{D2D}^{1,1}, r_{D2D}^{1,2}, ..., r_{D2D}^{1,m}\} \) denotes the D2D link rate between initiator \( m \) to collaborator \( i \). \( r_{D2D}^{m,n} = \{r_{D2D}^{1,m,1}, r_{D2D}^{1,m,2}, ..., r_{D2D}^{1,m,n}\} \) denotes the link rate at time slot \( t \) from initiator \( m \) to EN \( n \).

4.1.2 UE Action

After receiving the state \( s_t \), 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 \( a_t = \)
\{O_m, J_{m,n}, Z_{m,i}\} \in \{0, 1\}, \text{ where } O_m^t + J_{m,n}^t + Z_{m,i}^t = 1. \text{ 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 $s_t$ and action $a_t$, and guides the selection of the next action according to the reward $R_t$ 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 $R_t$ which can be expressed as $R_t = \sum_{m=1,t=1}^{M,T} \gamma^t \cdot r(s_m, a_t), \forall m \in M, \forall t \in T$, where the discount factor $\gamma \in [0, 1]$ demonstrates the importance of the instant reward. And $r(s_m, a_t) = D_m^t$ 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 $s_m^n$ at each time slot $t, t \in T$ (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 $Q(s_m^n; \theta_c)$ (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 Sec 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 $Q(s_t, a_t)$. The agent can observe the state $s_t$ from the environment, choose a course of action $a_t$ according to the policy $\pi(s_t, a_t)$, and generate an immediate reward $r_t$ (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 requests 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 $\theta^p$. 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: $\{H^t_a, ..., \hat{H}^t_k\} = g(H^t_a, ..., H^t_k)$. Then user $k$ communicates with $b$’s group: $\{\hat{H}^t_k, ..., \hat{H}^t_k\} = g(\hat{H}^t_k, ..., \hat{H}^t_k)$. 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.

Algorithm 1 Attention mechanism method

Input: Communication group $C$, a set of UE $m^p$

Output: initiator $m$, Communication group $C$ of the initiator

for episode $= 1, ..., \text{episode}_{\text{max}}$ do

for time slot $= 1, ..., T$ do

Get thought $H^t_m = \mu_1(s^t_m; \theta^s)$ for each initiator $m$

end

Select action intention $a^t_m = \mu_2(H^t_m, \hat{H}^t_m; \theta^p)$ for each agent $m$ without communication

end

Observation $s^t_m$ for each agent $m$ in communication group of initiator $m$

Select action $a^t_m = \mu_2(H^t_m, \theta^p)$ for each agent $m$ with communication

end

end

We train one critic network $\theta^c$ embedded with a attention mechanism for each initiator $m$. The integrated thought is defined as $\hat{H}^t_m$, it is merged with $H^t_m$, and input to the next attention mechanism policy network. Then the attention mechanism policy network outputs the action intention $a^t_m = \mu_2(H^t_m, \hat{H}^t_m; \theta^p)$. We calculate the average of the difference in Q-value between independent actions intention $a^t_i$ and cooperative actions:

$$\Delta Q^t_m = \frac{1}{|C|} \left( \sum_{i \in C} Q(s^t_i, a^t_i; \theta^p) - \sum_{i \in C} Q(s^t_i, \tilde{a}^t_i; \theta^p) \right)$$  (7)

We store $(\Delta Q^t_m, H^t_m)$ into a queue $\mathcal{U}$, and perform minmax normalization on $\Delta Q$ in $\mathcal{U}$. Then we will get $\Delta \tilde{Q} \in [0, 1]$, it can be used as the tag of the binary classification. We update $\theta^p$ in the loss function as:

$$L(\theta^p) = -\Delta \tilde{Q}^t_m \log(p(H^t_m; \theta^p)) + (1-\Delta \tilde{Q}^t_m) \log(1-p(H^t_m; \theta^p))$$  (8)

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

Considering the strategy improvement, we employ the Actor network $\mu(\theta)$ to generate the next state action $\mu(a_{t+1}|O_t)$ 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 $Q(\theta)$ to criticize the policy according to the estimated Q-value $Q(s_t, a_t|\theta^Q) \approx Q(s_t, a_t)$ by a method similar to supervised learning, so that the gradient can be written as (process 13 of Fig. 2):

$$\nabla_\theta Q(\mu) = E_{s_t,a_t,i \sim D}[\nabla_\theta Q(\mu(s_t, a_t)|a_\mu(s_t))]|_{a_\mu(s_t)} \tag{9}
$$

where $D$ is the experience replay memory containing $\{s_t, a_t, r_t, s_{t+1}, C\}$, and stores the transitions of all agents. The action value function can be updated as follow:

$$L(\theta) = E_{s_t,a_t,r_t,s_{t+1},C}[(Q(\mu(s_t, a_t)) - y)^2] \tag{10}
$$

where $y = r + \gamma Q^\mu(s_{t+1}, a_{t+1})|a_\mu(s_{t+1})$.

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

$$\nabla_\theta \mu = \nabla_a Q(s_t, a_t|\theta^Q)|_{s_t=a_t=\mu(s_t)} \tag{11}
$$

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 $Q(s, a|\theta^Q)$ is also used to calculate the target value $r_t + \gamma Q^\mu(s_{t+1}, a_{t+1}|\mu(s_{t+1})|\theta^\mu)$, 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, $\theta^\prime \leftarrow \tau \theta^\prime + (1-\tau)\theta$, $\tau \ll 1$.

The MEC environment receives and performs the offloading action $a_t$ returned by the Agent during the learning process (process 16 of Fig. 2). Then, the Agent feeds back a reward $r_t$ and a next state $s_{t+1}$. Normally, an experience replay mechanism is used in this process which stores previous experiences $\{s_t, a_t, r_t, s_{t+1}, C\}$, then sampling these experiences evenly in small batches $D^\prime$, 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) $\psi$ is recorded as Q-value, reflecting the degree to which Agent has learned from current experience. It can be expressed as follow:

$$\psi = r_t + \gamma \max_{a_{t+1}} Q^\mu(s_{t+1}, a_{t+1}) - Q^\mu(s_t, a_t)|_{a_\mu(s_t)} \tag{12}
$$

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 $\psi$, we define the sampling probability $P_m$ of Agent $m$ as $P_m = \frac{\epsilon_m}{\sum_j \psi_j}$, where $\alpha$ is the control parameters for sorting quantity for priority, $j$ is the index of the PER batch sample, $p_m = |\psi_m| + \epsilon$, and $\epsilon$ 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 $\beta$ is to adjust the correction degree. In the process of Q-value learning update, $\eta_m \psi_m$ is used instead of $\psi_m$. 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:

**Algorithm 1 D2D-assisted Attention Cooperative Multi-agent Deep Reinforcement Learning**

**Input:** $\theta^\mu, \theta^\phi, \theta^\mu, \theta^\phi, \theta^Q, \theta^\phi$ and $\theta^P$ for attention cooperative formulation, $\gamma$ for the target network, $\alpha$ and $\beta$ for PER method.

**Output:** Computation offloading decision for Agent $m \in M$

for $t = 1$ to $episode_{max}$ do

| Choose action $a_t \sim \mu_\theta(a_t|s_t)$ for each Agent.
| 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 $D$;
| Update $s \leftarrow s_{t+1}$;
| if $t + 1 \mod T = 0$ then
| Sample a subset $D’$ from $D$ by PER
| Set $y = r + \gamma Q^\mu(s_{t+1}, a_{t+1}|a_\mu(s_{t+1})$ 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 $\theta^Q \leftarrow \theta^Q + \alpha_\theta \cdot \nabla_\theta L^Q(\theta^Q)$;
| Update the Actor network using $\theta^\mu \leftarrow \theta^\mu + \alpha_\theta \cdot \theta^\mu$;
| Update the target networks using (14);

end

end

end
Critic networks (process 9 and process 14 of Fig. 2), where $\alpha^Q$ and $\alpha^\mu$ are the network learning rates:

$$\theta^Q \leftarrow \theta^Q + \alpha^Q \cdot \nabla_{\theta^Q} L^Q(\theta^Q), \quad \theta^\mu \leftarrow \theta^\mu + \alpha^\mu \cdot \nabla_{\theta^\mu}$$  \hspace{1cm} (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):

$$\theta^{Q,\tau} \leftarrow \tau \theta^Q + (1 - \tau) \theta^{Q,\tau'}, \quad \theta^{\mu,\tau} \leftarrow \tau \theta^\mu + (1 - \tau) \theta^{\mu,\tau'}$$  \hspace{1cm} (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 $T$ time steps in the reinforcement learning algorithm, i.e., $T = \{1, 2, ..., t\}$, the computational complexity can be simplified expressed as $O(T^2)$ [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 $O(I^2)$. For the attention mechanism, the complexity is $O(P^2 \cdot Q)$ according to Algorithm 1. Moreover, assume that the computation complexity for the training of one experience is $O(L)$, where $L$ is the number of multiplication operations in the neural network, the computation complexity of the proposed algorithm is $O(LP^2QV)$ 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.

#### 5.2 Convergence analysis

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

5.2.1 Episode vs. 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.

Fig. 3 (a) 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. 3 (b) 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.

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

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 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 UE $M^p$, 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 increases 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

![Fig. 4. Convergence time and execution time as the number of UEs varies.](image)

<table>
<thead>
<tr>
<th>$M^p$</th>
<th>20</th>
<th>40</th>
<th>60</th>
<th>80</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>10%</td>
<td>0.3933</td>
<td>0.3998</td>
<td>0.4619</td>
<td>0.5167</td>
<td>0.5494</td>
</tr>
<tr>
<td>30%</td>
<td>0.3806</td>
<td>0.4034</td>
<td>0.4614</td>
<td>0.5114</td>
<td>0.5312</td>
</tr>
<tr>
<td>50%</td>
<td>0.3812</td>
<td>0.4335</td>
<td>0.4767</td>
<td>0.5237</td>
<td>0.5563</td>
</tr>
</tbody>
</table>

![Normalized Average System Reward](image)

**TABLE 3**
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 algorithm can be regarded as a single agent to make the computation offloading dependently. For the MADRL, we adopt the state-of-the-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. 5 (a), 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. 5 (b), 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. 6 (a), 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. 6 (b), 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.

### 6 Conclusion

In this paper, 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.

### ACKNOWLEDGMENTS

This work is supported by the National Key R&D Program of China under Grant No. 2022YFB4400800; the National Natural Science Foundation of China under Grant No. 62032013 and No. 61872073; the LiaoNing Revitalization Talents Program under Grant No. XLYC1902010.

### REFERENCES


Kexin Li received the B.S. degree in software engineering from Harbin University of Science and Technology, Harbin, China, in 2015, and the M.S. degree in computer technology from Northeastern University, Shenyang, China, in 2018, where she is currently pursuing the Ph.D. degree in computer application technology.
Her research interests include software-defined networking, edge computing, and machine learning.

Xingwei Wang received the B.S., M.S., and Ph.D. 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. Prof. Wang has received several best paper awards.

Qiang He received the Ph.D. 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 Ph.D. Researcher from 2018 to 2019. He has published more than ten journal articles and conference papers.
His research interests include social network analysis, machine learning, data mining, and software-defined networking.

Mingzhou Yang received the B.Sc. degree in computer science and technology from Shaneng University of Technology, Shenyang, China, in 2015, and the M.Sc. degree in computer software and theory from Northeastern University, Shenyang, China, in 2017. She is currently working toward the Ph.D. degree in computer application technology from Northeastern University, Shenyang, China.
Her research interests include social network analysis and computational intelligence.

Min Huang (Member, IEEE) received the B.S. degree in automatic instrument, the M.S. degree in systems engineering, and the Ph.D. 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.