On Jointly Optimizing Partial Offloading and SFC Mapping: A Cooperative Dual-Agent Deep Reinforcement Learning Approach

Xinhua Wang, Huanlai Xing, Member, IEEE, Fuhong Song, Shouxi Luo, Member, IEEE, Penglin Dai, Member, IEEE, and Bowen Zhao

Abstract—Multi-access edge computing (MEC) and network function virtualization (NFV) are promising technologies to support emerging IoT applications, especially those computation-intensive. In NFV-enabled MEC environment, service function chain (SFC), i.e., a set of ordered virtual network functions (VNFs), can be mapped on MEC servers. Mobile devices (MDs) can offload computation-intensive applications, which can be represented by SFCs, fully or partially to MEC servers for remote execution. This article studies the partial offloading and SFC mapping joint optimization (POSMJO) problem in an NFV-enabled MEC system, where the data from an incoming task is partitioned into two parts, with one part executed locally and the other offloaded to the edge infrastructure for execution. These two parts are independent of each other, but both need to be processed by the same SFC. The objective is to minimize the average cost in the long term which is a combination of execution delay, MD’s energy consumption, and usage charge for edge computing. This problem consists of two closely related decision-making steps, namely task partition and VNF placement, which is highly complex and quite challenging. To address this, we propose a cooperative dual-agent deep reinforcement learning (CDADRL) algorithm, where two agents interact with each other. Simulation results show that the proposed algorithm outperforms three combinations of deep reinforcement learning algorithms with respect to cumulative reward and it overweighs a number of baseline algorithms in terms of execution delay, energy consumption, and usage charge.

Index Terms—Deep reinforcement learning, multi-access edge computing, network function virtualization, partial offloading, SFC mapping.

I. INTRODUCTION

Due to the advent of fifth generation (5G) networks and Internet of Things (IoT) era, the number of mobile devices (MDs), including smartphones, wearables, tablets, sensors, etc., has increased rapidly [1], [2]. Meanwhile, many computation-intensive IoT applications are emerging, such as augmented/virtual reality (AR/VR), interactive online gaming and object recognition. Such applications are usually highly dependent on the availability of computing and storage resources. When executing these applications, MDs are strictly constrained by limited computing, communication, and battery capacities [3]. Multi-access edge computing (MEC) has been considered as a promising solution to the problem above as computation offloading allows MDs to offload computation-intensive applications to MEC servers for execution, reducing the computing pressure on MDs [4], [5].

In MEC, computation offloading can be classified into two categories, namely binary and partial offloading [3]. In binary offloading, a task is executed either locally or remotely, i.e., “all or none”. In contrast, partial offloading partitions a task into two parts. One part is run on the MD and the other part is offloaded to the edge infrastructure (EI) for remote execution. Both parts can be executed in parallel, which helps to reduce application latency and energy consumption, thus significantly improves the quality of experience (QoE) [6]. Therefore, this paper focuses on the partial offloading model.

On the other hand, network function virtualization (NFV) has been envisaged as one of the fundamental technologies for next-generation networks, e.g., the sixth generation (6G) network and space-air-ground integrated network (SAGIN). NFV decouples network functions from proprietary hardware and implements them as virtual network functions (VNFs). Faced with the explosion of computation-intensive and delay-sensitive applications, enabling NFV in MEC ensures users’ QoE and reduces both capital expenditure (CAPEX) and operational expense (OPEX) from the point of view of network service providers (NSPs), which is an inevitable trend of MEC [7]. The European Telecommunications Standards Institute (ETSI) introduced an MEC reference architecture in [8], where MEC was deployed as part of an NFV environment. In this architecture, a mobile edge application is represented by a set of ordered VNFs called service function chain (SFC) [9]. Considering that a single MEC
server may not be capable of running all the VNFs required due to its limited hardware resources. NSPs prefer to employ multiple MEC servers for load balancing and QoE-guarantee purpose [10].

In the literature, most studies on NFV-enabled MEC systems focused on SFC mapping in EI environment. They considered placing VNFs on MEC servers in order to optimize the end-to-end delay, energy consumption or other performance metrics. Recently, jointly optimizing computation offloading and SFC mapping has attracted increasingly more research interests. Most of them, however, consider binary offloading. With respect to partial offloading, the data from an incoming task can be partitioned into two parts, each to be executed in parallel on the MD and at the EI. The streaming pipeline pattern is considered when processing an arbitrary task, where a pipeline of VNFs is invoked in a sequential fashion [11]. NSP instantiates these VNFs on the MD and at the edge, steering data flows through them one after another. The decision making for partial offloading and that for the corresponding VNF placement are closely related [12], [13]. Provisioning partial offloading service in NFV-enabled MEC network with a specific SFC requirement poses many challenges. For example, how to partition the task to achieve optimal offloading performance? How to place the required VNFs on proper MEC servers to meet service demands? And how to jointly optimize the partial offloading and SFC mapping in order to find an acceptable trade-off among execution delay, energy consumption and usage charge?

The partial offloading and SFC mapping joint optimization (POSMJO) consists of two closely coupled decision-making steps, namely task partition and VNF placement. The task partition step decides the proportion of data to be offloaded for remote execution, while the VNF placement step deploys the corresponding VNFs on appropriate MEC servers. Both of them are NP-hard yet require realtime solution. As many applications have short execution deadlines, such as VR/AR, the MD needs to quickly make offloading and VNF placement decisions [5]. Deep reinforcement learning (DRL) algorithms, which combine reinforcement learning (RL) with deep neural network (DNN), represent promising approaches to handling NP-hard problems like computation offloading [5] and SFC mapping [14]. DRL can quickly make decisions after training, thus satisfying the requirements of delay-sensitive applications. On the other hand, task partition is a prerequisite for VNF placement, and the result of VNF placement directly affects the performance of task partition. Thus, the two steps are complementary and closely related. To obtain an acceptable solution to the POSMJO problem demands for appropriate collaboration between them. Task partition and VNF placement are continuous and discrete optimization problems, respectively. Traditional DRL can handle either environments with discrete actions, e.g., deep Q-network (DQN) [15], or those with continuous actions, e.g., deep deterministic policy gradient (DDPG) [16]. However, using a single-agent DRL to tackle the two steps above simultaneously is almost impossible. This motivates us to design a cooperative dual-agent DRL (CDADRL) algorithm to address the POSMJO problem, where two agents collaborate on dealing with the two closely related steps.

This paper studies the POSMJO problem in the NFV-enabled MEC environment, with data-partitioned-oriented tasks [6] considered. Unlike existing studies focusing on binary offloading, the data from an incoming task is partitioned into two parts, with one part executed locally and the other offloaded to the edge infrastructure for execution. These two parts are independent of each other, but both need to be processed by the same SFC. For instance, for data cleaning, the data can be arbitrarily partitioned into local and remote execution parts, both of which go through a series of atomic functions, such as dataset merging, missing data rebuilding, standardization, normalization, and de-duplication. After local and remote execution, data integration is performed on the MD to output the cleaned data. A DRL algorithm with two collaborative agents is presented to address the problem. The main contributions of this paper are listed as follows.

- We formulate the POSMJO problem in the context of NFV-enabled MEC, where the execution delay, energy consumption, and usage charge are three objectives to minimize, subject to a number of constraints such as computing and bandwidth capacities, and task demand. The problem consists of two closely related decision-making steps, namely task partition and VNF placement.
- A CDADRL algorithm is devised to address the POSMJO problem, where a framework with two environments, namely, the MD and EI environments, is used to support the interaction between two agents. One agent interacts with the MD environment for task partition, while the other interacts with the EI environment for VNF placement.
- We conduct extensive simulations to evaluate the performance of the proposed algorithm. The results demonstrate that CDADRL outperforms three combinations of state-of-the-art DRL algorithms in terms of cumulative reward. In addition, the proposed algorithm obtains lower execution delay, energy consumption and usage charge values, compared with six baseline algorithms.

The rest of the paper is organized as follows. The related work is reviewed in Section II. Section III describes the system model and problem formulation. Then, Section IV introduces in detail the proposed CDADRL algorithm for the POSMJO problem. Section V evaluates the performance of the proposed algorithm. Section VI concludes the paper and highlights potential avenues for future research.

II. RELATED WORK

Over the past few years, there have been increased efforts to study NFV-enabled MEC. Generally speaking, these studies can be classified into two categories, namely VNF placement optimization and computation offloading and VNF placement joint optimization.

A. VNF Placement Optimization

In NFV-enabled MEC systems, mobile edge applications are implemented as SFCs in the EI environment, where each SFC is a set of ordered VNFs deployed on a substrate network [8]. Huang et al. [17] considered the reliability-aware VNF placement in MEC and formulated it as an integer linear programming (ILP),
where an approximation algorithm was proposed to maximize the network throughput. Chantre et al. [18] studied the location problem for the provisioning of protected slices in NFV-enabled MEC. They modeled it as a multi-criteria optimization problem and proposed a multi-objective evolutionary non-dominated sorting genetic algorithm. Subramanya et al. [19] converted the SFC placement problem in an MEC-enabled 5G network into an ILP model, where machine learning methods were adopted for network traffic prediction. Zheng et al. [20] studied a hybrid SFC embedding problem in the context of MEC and addressed it by a heuristic algorithm based on hybrid SFC embedding auxiliary graph. Yang et al. [21] investigated the VNF placement and routing problem in edge clouds and proposed a randomized rounding approximation algorithm to minimize the maximum link load. Xu et al. [22] studied the provisioning of IoT applications in NFV-enabled MEC, where the IoT application and VNFs of each service request were consolidated into a single location. They devised a heuristic algorithm to jointly place the IoT application and its corresponding VNFs. Ma et al. [23] considered mobility-aware and delay-sensitive service provisioning in MEC, where an approximation algorithm was devised to maximize the network utility and throughput.

Apart from the methods above, DRL-based approaches exhibit a rising trend. Shu et al. [24] introduced the problem of deploying VNFs in MEC with time-varying geographical of users, where a DRL-based method was employed to estimate the underlying wireless features affected by user mobility and improve the resource utilization of MEC servers. Dalgkitis et al. [25] emphasized the automatic VNF placement problem in 5G network, where DDPG was used to minimize latency for ultra-reliable low-latency communication (uRLLC) services. Liu et al. [26] investigated SFC orchestration problem in NFV/MEC-enabled IoT networks, where a DRL method was devised for dynamic SFC orchestration. Later on, the authors improved their work by leveraging DDPG and asynchronous advantage actor-critic (A3C) that handled small- and large-scale networks, respectively [14]. Xu et al. [27] focused on SFC mapping for industrial IoT (IIoT), where a DQN-based online SFC deployment method was designed to balance the quality of IIoT services and resource consumption. Abouaomar et al. [28] investigated the resource provisioning problem for VNF placement and chaining in MEC. The authors proposed a mean-field game framework to model the behavior of placing and chaining VNFs, and they leveraged an actor-critic DRL to learn the optimal placement and chaining policies.

B. Computation Offloading and VNF Placement Joint Optimization

Computation offloading is an effective way for 5G/6G networks to deal with computation-intensive and delay-sensitive tasks. In partial offloading, for an arbitrary task, the part for local execution and that for remote execution need to be processed by two identical SFCs, respectively. Therefore, the decision making process of computation offloading and that of VNF placement are closely related, forming a joint optimization problem. This problem has attracted increasingly more research attention from both academia and industry. For instance, Ma et al. [7] jointly optimized VNF placement and user requests assignment in MEC. If the computing capacity on each cloudlet was sufficient, an ILP solution was used to maximize the requests admission number; otherwise, two heuristic algorithms were adopted to maximize the network throughput. Nguyen et al. [9] investigated the joint optimization problem of computation offloading, SFC placement and resource allocation in NFV-enabled MEC, where a decomposition approach was employed to minimize the weighted normalized energy consumption and computing cost. Kim et al. [12] designed a dual-resource NFV-enabled MEC system, with the multi-path and sending rate control problem studied. A convex optimization method based on Lagrange dual theory was developed to handle the problem constraints, and an extragradient-based algorithm was proposed to control and split the service sending rate. Jia et al. [10] formulated a task offloading and VNF placement problem in a metropolitan area network, where each offloaded task requested a specific VNF. The authors presented an online heuristic algorithm to maximize the requests admission number. Xu et al. [29] investigated the task offloading problem in MEC with quality of service (QoS) requirement and devised an online algorithm with competitive ratio. Jin et al. [30] emphasized computation offloading of online games in a fog-enabled heterogeneous radio access network, where the games’ interactive and uncertain features were modeled as probabilistic SFCs. They formulated the computation offloading problem as an ILP and proposed a heuristic algorithm to address it. Khoramnejad et al. [13] looked into the partial offloading problem in NFV-enabled MEC, where each VNF in the required SFC could be executed locally or offloaded to an MEC server. The authors adapted double DQN (DDQN) to the problem and obtained decent performance.

Besides, some studies investigated task offloading and resource allocation joint optimization problems in MEC network. Khoramnejad et al. [31] studied the problem of joint offloading and resource allocation in MEC-enabled networks, where a DRL-based algorithm was proposed to maximize the number of computed bits and minimize the energy consumption. Zhang et al. [32] jointly optimized the cooperative partial task offloading and resource allocation problem for IIoT applications. The authors designed an improved soft actor-critic-based method to minimize the long-term total system costs, with the task processing delay and energy consumption constraints satisfied. Zhou et al. [33] investigated the joint optimization problem of computation offloading, service caching, and resource allocation in a collaborative MEC system. They formulated the problem as mixed-integer non-linear programming and proposed a DDPG to minimize the long-term energy consumption.

Table I clearly shows the differences between similar works and ours in terms of five aspects, namely, network scenario, system model, offloading model, offloading decision, and optimization technique. In this paper, we consider POSMJO in the context of NFV-enabled MEC. We distinguish our work from the existing ones as follows.

- In terms of computation offloading model, most existing works focus on binary offloading [7], [9], [10], [12], [29], [30]. However, some applications contain some
non-offloadable part(s), e.g., user input, camera, and positioning request, that need to be executed on the MD [34]. Therefore, binary offloading cannot be applied to these applications. Except [13], none of the studies above considered partial offloading. However, ref. [13] only considered where to execute each given VNF, the MD or a designated MEC server. Whether task data could be partitioned for local and edge execution was not considered. With the advent of 5G networks and imminent of 6G networks, it is inevitable to provide partial offloading in NFV-enabled MEC, making full use of the computation offloading advantages. Our work assumes that the data from an incoming task is arbitrarily partitioned into two parts that go through two identical SFCs, separately. One part is executed on the local MD and the other at the edge in parallel.

- Most existing computation offloading and SFC mapping methods are based on heuristic [7], [9], [10], [20], [22], [29], [30], approximation [17], [21], [23], convex optimization [12], and meta-heuristic algorithms [18]. However, these methods are usually highly complex and time-consuming, which are not suitable for dynamic MEC scenarios. On the other hand, DRL can predict environmental information through offline model training and make near-optimal realtime decisions, making it especially suitable for dynamic and complex environments. Therefore, it is natural to adapt DRL for the POSMJO problem. DRL-based methods [13], [14], [24], [25], [26], [27], [31], [32], [33] have demonstrated their ability for efficiently handling various decision making processes in time-varying MEC systems. In [13], the proposed DRL algorithm was used to decide whether a VNF was executed locally or at the edge. Refs. [14], [24], [25], [26], [27] applied DRL methods to VNF placement in MEC networks. Refs. [31], [32], [33] investigated the task offloading and resource allocation joint optimization problem in MEC network and addressed the problem based on DRL methods. In NFV-enabled MEC, task partition and VNF placement are continuous and discrete optimization problems, respectively, and they are closely related. However, the above works did not consider such a problem, making them not directly applicable to solving the POSMJO problem. We consider the relation between the two closely related decision-making steps, namely task partition and VNF placement, and we devise a DRL algorithm with two cooperative agents to handle the POSMJO problem.

### III. System Model and Problem Formulation

We consider an NFV-enabled MEC system consisting of one MD and the edge infrastructure (EI). The EI is represented by an undirected graph G = (V, E), where V is the set of base stations (BSs) and E is the set of wired links between BSs. Each v ∈ V is connected to an MEC server with limited computing capacity. Each MEC server can host multiple VNFs to implement the required mobile edge applications and provide computation offloading services to the MD. We consider a discrete-time system where t ∈ {1, ..., T} is the t-th time slot and T is the monitoring period in terms of numbers of time slots. Let cp_v^t represent the available computing capacity of BS v ∈ V in time slot t. Denote the available bandwidth capacity of link e ∈ E in time slot t by be_e^t. In the system, the BS connected with the most powerful MEC server is designated as the central management node that runs the NFV management and orchestration (NFV-MANO) framework [8]. The NFV-MANO is responsible for VNF orchestration and life cycle management, including VNF placement, routing, and so on.

The MD communicates with the EI through the closest BS v_{MD} to receive computation offloading services [35]. In partial offloading, the input data of a task can be arbitrarily partitioned into two parts for local and edge computing [6], [36], [37]. The two parts are processed by two identical SFCs, one deployed on the MD and the other at the edge [9].

| References | [7] | [9] | [12] | [13] | [14] | [17] | [18] | [19] | [20] | [21] | [22] | [23] |
|------------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| Network scenario | Pure MEC | NFV-enabled MEC | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| System model | VNF placement | Joint optimization | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Offloading model | Binary | Partial | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Offloading decision | Data partition | VNF placement | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Optimization technique | Traditional | DRL | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

| References | [24] | [25] | [26] | [27] | [28] | [10] | [29] | [30] | [31] | [32] | [33] | Ours |
|------------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| Network scenario | Pure MEC | NFV-enabled MEC | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| System model | VNF placement | Joint optimization | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Offloading model | Binary | Partial | N/A | N/A | N/A | N/A | N/A | ✓ | ✓ | ✓ | ✓ | ✓ |
| Offloading decision | Data partition | VNF placement | N/A | N/A | N/A | N/A | N/A | ✓ | ✓ | ✓ | ✓ | ✓ |
| Optimization technique | Traditional | DRL | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
$u^t = \{ F^t, d^t, c^t, D^t_{\text{max}} \}$ be the task of the MD in time slot $t$, where $F^t$ represents the required SFC, $d^t$ is the input data size, $c^t$ is the number of CPU cycles required to execute $u^t$, and $D^t_{\text{max}}$ is the execution deadline of $u^t$. Let $f_{i}^t$, $i = 1, \ldots, N^t$, be the $i$-th VNF in $F^t$ and $cp(f_{i}^t)$ be the computing capacity consumed for implementing $f_{i}^t$, where $N^t$ is the number of VNFs in $F^t$. Assume the delay for instantiating $f_{i}^t$ on the MD or an MEC server is $D(f_{i}^t)$. The required bandwidth capacity between $f_{i}^t$ and $f_{i+1}^t$ is denoted by $bw(f_{i}^t)$. The data flow of $u^t$ needs to be processed by $f_{i}^t, \ldots, f_{N^t}^t$. Let $x^t \in [0, 1]$ denote the offloading ratio for partitioning $u^t$, indicating how much data in $d^t$ is offloaded to the edge, i.e., $x^t \cdot d^t$ is the amount of data to be executed remotely. Let $h_{i}^t$, $i = 1, \ldots, N^t$, be the amount of output data after $u^t$ is processed by $f_{i}^t$. Let $\xi_{i}^t$ be the ratio of $h_{i}^t$ to $d^t$. $\xi_{i}^t \cdot d^t$ is the amount of output data after $u^t$ is processed by $f_{i}^t$. This paper assumes that the data from an incoming task is partitioned into two independent parts. One part is executed locally while the other is offloaded to the edge infrastructure for execution. Both of them need to be processed by the same SFC. In other words, each VNF instance in SFC $F^t$ required by task $u^t$ is instantiated on the MD and one selected MEC server so that the two parts above can be executed on the MD and at the edge in parallel. Fig. 1 depicts an example of joint partial offloading and SFC mapping in the NFV-enabled MEC scenario. There are seven BSs in the EI and the NFV-MANO framework is running on BS4. There is an MD connecting to BS1 and an incoming task $u^t$ arrives in time slot $t$. The input data of task $u^t$ goes through $VNF1 \rightarrow VNF2 \rightarrow VNF3$ to implement some mobile edge application. Assume the MD partitions the input of $u^t$ into two parts with $x^t = 0.6$, i.e., 40% of the input task data is processed on the MD while the rest is offloaded to the EI for remote execution. The NFV-MANO selects BS2, BS6, and BS7 to host these VNFs, and at the same time the MD instantiates the required VNFs locally subject to the availability of its computing resources. The offloaded data is processed at the EI and the result is delivered back to the MD. The MD integrates the two parts and then task $u^t$ is complete.

According to the Shannon-Hartley theorem [38], the achievable uplink transmission rate from the MD to BS $v_{MD}^{t}$ is

$$u^t_p = W \cdot \log_2 \left( 1 + \frac{p_{MD}^{t} \cdot g_{up}^{t}}{\eta^2} \right),$$

where $W$, $p_{MD}^{t}$, and $\eta^2$ are the wireless bandwidth, transmission power of the MD, and noise power, respectively. $g_{up}^{t}$ is the uplink channel gain between the MD and $v_{MD}^{t}$. Similarly, the achievable downlink transmission rate is

$$down^t = W \cdot \log_2 \left( 1 + \frac{p_{v_{MD}^{t}}^{t} \cdot g_{down}^{t}}{\eta^2} \right),$$

where $p_{v_{MD}^{t}}^{t}$ is the transmission power of BS $v_{MD}^{t}$, and $g_{down}^{t}$ is the downlink channel gain between $v_{MD}^{t}$ and the MD. The main notations used in this paper are summarized in Table II.

### A. Local Computing

Note that we may not instantiate all VNFs in $F^t$ simultaneously on the MD, due to the limitation of its computing capacity. We divide $F^t$ into $\psi^t$ groups according to the MD’s computing capacity $cp_{MD}^{t}$ in time slot $t$. To make full use of the MD’s computing capacity, the MD releases its corresponding computing resources occupied immediately after the input data has been processed by all VNFs in the $k$-th group $F_k^t$ orderly. The relation between $F^t$ and $F_k^t$ is defined as

$$\bigcup_{k=1}^{\psi^t} F_k^t = F^t.$$

In time slot $t$, the computing capacity consumed by $F_k^t$ cannot exceed the MD’s computing capacity $cp_{MD}^{t}$, i.e.,

$$\sum_{j=1}^{N_k^t} cp(f_{k,j}^t) \leq cp_{MD}^{t},$$

where $f_{k,j}^t \in F_k^t$ is the $j$-th VNF in $F_k^t$, and $N_k^t$ is the number of VNFs in $F_k^t$. Take Fig. 1 as an example. If the MD’s computing capacity can only afford to host $VNF1$ and $VNF2$ at the same time, we instantiate them at first. After the task data is processed by them, the MD releases the computing resources occupied by $VNF1$ and $VNF2$. And then it instantiates $VNF3$ to complete the local execution of $u^t$. After all VNFs in $F_k^t$ are instantiated, the amount of input data of $u^t$, $(1 - x^t) \cdot d^t$, is processed by every VNF in $F_k^t$ in the right order. That $f_{k,j}^t$ processes the task incurs a processing
the local computing delay, $DL_k$, we calculate the local computing delay, $DL_k$, by

$$DL_k = \sum_{i=1}^{N_k} \left( DI(F_{k,i}) + \sum_{j=1}^{N_k} DF_{k,j} \right).$$  

(6)

The energy consumption for local computing, $EL_k$, is defined as [39]

$$EL_k = \sum_{i=1}^{N_k} \xi_{k,i} \cdot (1 - \chi) \cdot \eta \cdot \kappa \cdot \left( cp(f_i) \right)^2,$$  

(7)

where $\kappa$ is the effective capacitance coefficient dependent on the chip architecture used and $\xi_{k,i} = 1$. $\xi_{k,i} = (1 - \chi) \cdot \eta \cdot \kappa$ is the amount of data to be processed by $f_i$ locally. Thus, $\xi_{k,i} \cdot (1 - \chi) \cdot \eta \cdot \kappa$ is the number of CPU cycles required to execute $f_i$ over $f_i$. $cp(f_i)$ is the required computing capacity (CPU frequency) for executing $f_i$. According to [39], the power consumption of the CPU is approximately proportional to the product of the circuit supplied voltage and $cp(f_i)^2$. Besides, when the CPU works at the low-voltage limit, the corresponding circuit supplied voltage is approximately linearly related to the CPU frequency. Therefore, the energy consumption per CPU cycle is given by $\kappa \cdot \left( cp(f_i) \right)^2$.

Note that, energy consumption is affected by the utilization of computers. Especially in a performance interference model, as the utilization rate changes, the energy consumption will change over time. In NFV-enabled MEC, VNFs are implemented in virtual machine (VM) or container environment [40], indicating that each VNF instance running on the MD or MEC server is isolated [41]. Besides, existing studies have been conducted to dynamically scale up/down Docker resources according to thresholds calculated by monitored metrics, so as to ensure the execution performance of applications. For example, the Docker controller can dynamically adjust memory, CPU time, vCPU core according to workloads and thresholds [42]. Therefore, this paper assumes that energy consumption is not affected by the utilization of computers.

### B. Edge Computing

In time slot $t$, a proportion of the input data of $u^t$, $x^t \cdot d^t$, needs to be offloaded to the EL for remote execution. At first, NFV-MANO generates an SFC mapping scheme for $F^t$ according to the MD’s offloading decision and issues relevant instructions to the corresponding BSs. Then, these BSs instantiate the requested VNFs. Denote by $\rho_t$ the BS selected to host VNF $f_i \in F^t$. The instantiation delay of $F^t$ at the EL, $DI(F^t)$, is defined as $DI(F^t) = \max \{|DI(f_i)|; \forall f_i \in F^t\}$. Meanwhile, considering that any two BSs in the system are close to each other, this paper ignores the propagation delay.

The transmission delay of task $u^t$ consists of three parts: (1) the data transmitted through the wireless uplink channel from the MD to BS $v_{MD}^t$, (2) the data transmitted through wired links between BSs, and (3) the data transmitted through the wireless downlink channel from BS $v_{MD}^t$ to the MD. Thus, the total delay, $DP_{k,j}^t$, written as

$$DP_{k,j}^t = \begin{cases} \chi_{k,j} \cdot \chi_{k,j-1}, & \text{if } k = 1, j = 1 \\ \chi_{k,j} \cdot \chi_{k,j-1}, & \text{if } k = 2, \ldots, \psi, j = 1 \\ \chi_{k,j} \cdot \chi_{k,j-1}, & \text{if } k = 1, \psi, j = 2, \ldots, N_k \\ \chi_{k,j} \cdot \chi_{k,j-1}, & \text{otherwise} \end{cases}$$  

(5)

where $\chi = (1 - \chi) \cdot \eta \cdot \kappa$, and $\chi_{k,j}$ is the ratio of $h_{k,j}^t$ to $d^t$, and $h_{k,j}^t$ is the amount of output data after $u^t$ is processed by $f_{k,j}^t$. Let $DI(F_k^t)$ be the instantiation delay of $F_k^t$, where $DI(F_k^t)$ is equal to the maximum VNF instantiation delay in $F_k^t$, i.e., $DI(F_k^t) = \max \{|DI(f_i)|; \forall f_i \in F_k^t\}$. Based on

| Notation | Definition |
|----------|------------|
| $b(u^t_{f_i})$ | Required bandwidth capacity between $f_i^t$ and $f_{i+1}^t$ |
| $b^t$ | Available bandwidth capacity of e at t |
| $c^t$ | Number of CPU cycles required to execute $u^t$ |
| $cp(f_i^t)$ | Required computing capacity for execution $f_i^t$ |
| $cp_{MD}^t$ | Available computing capacity of the MD at t |
| $d^t$ | Input data size of $u^t$ |
| $D_k^t$ | Execution deadline of $u^t$ |
| $DC_k$ | Execution delay of $u^t$ |
| $DE_k$ | Edge computing delay |
| $DI(F_k^t)$ | Instantiation Delay of $F_k^t$ |
| $DL_k^t$ | Local computing delay |
| $DT_k^t$ | Transmission delay of $u^t$ |
| $down_k^t$ | Achievable downlink transmission rate |
| $EC_k^t$ | Energy consumption of the MD for executing $u^t$ |
| $EP_k^t$ | Energy consumption for edge computing |
| $EL_k^t$ | Energy consumption for local computing |
| $EU_k^t$ | Energy consumption for transmitting data from the MD to $v_{MD}^t$ |
| $F_k^t$ | Required SFC corresponding to $u^t$ |
| $F^t$ | The k-th group of $F^t$ |
| $f_i^t$ | The i-th VNF in $F^t$ |
| $f_{k,i}^t$ | The j-th VNF in $F_k^t$ |
| $G = (V,E)$ | Edge infrastructure |
| $g_{down}^t$ | Downlink channel gain between $v_{MD}^t$ and the MD |
| $g_{up}^t$ | Uplink channel gain between the MD and $v_{MD}^t$ |
| $h_k^t$ | The amount of output data after $u^t$ is processed by $f_i^t$ |
| $N_k^t$ | Number of VNFs in $F_k^t$ |
| $N_{up}$ | Number of VNFs in $F^t$ |
| $p_{MD}^t$ | Transmission power of the MD |
| $p_{MD}^t$ | Receiving power of the MD |
| $p_{MD}^t$ | Transmission power of $v_{MD}^t$ |
| $\rho_t$ | BS selected to host VNF $f_i^t \in F^t$ |
| $UC_k^t$ | Usage charge for edge computing |
| $u^t$ | Incoming task of the MD at t |
| $up_k^t$ | Achievable uplink transmission rate |
| $v_{MD}^t$ | The closest BS to the MD |
| $W$ | Wireless bandwidth |
| $x^t$ | Offloading ratio |
| $\xi_{k,i}$ | Ratio of $h_{k,j}^t$ to $d^t$ |
| $\eta_{k,i}$ | Noise power |

Note that, $\chi$ is written in reinforcement learning. 

$a$ | Action |
| $A$ | Action space |
| $A_o$ | Agent in charge of task partition |
| $A_p$ | Agent in charge of VNF placement |
| $B$ | Replay buffer |
| $r(s,a)$ | Reward function |
| $S$ | State |
| $\gamma$ | Discount factor |
transmission delay of task $u^t$ is defined as

\[ DT^t = \frac{x^t \cdot d^t}{u^t \cdot \rho^t} + \sum_{e \in E_1} \frac{x^t \cdot d^t}{bw_e^t} + \sum_{i=1}^{N^t-1} \frac{\xi_{i-1} \cdot x^t \cdot d^t}{bw(f_i^t)} + \sum_{e \in E_N^t} \frac{\xi_{N^t-1} \cdot x^t \cdot d^t}{bw_e^t}, \]

where $E_1^t$ is the link set of the path from $v_{MD}^t$ to the BS hosting $f_1^t$, and $E_N^t$ is the link set of the path from the BS hosting $f_{N^t}^t$ to BS $v_{MD}^t$. Note that the transmission path between BSs involved in this paper is generated by the shortest path method, e.g., Dijkstra's algorithm [43].

The total processing delay incurred by VNFs $f_1^t, \ldots, f_{N^t}^t$ processing the offloaded data at the EI, $DP_E^t$, is defined by

\[ DP_E^t = \sum_{i=1}^{N^t} \frac{\xi_{i-1} \cdot x^t \cdot d^t \cdot c^t}{cp(f_i^t)}. \]

Based on the instantiation, transmission and processing delays, $DI(f_i^t)$, $DT^t$ and $DP_E^t$, we calculate the edge computing delay, $DE^t$, by

\[ DE^t = DI(f_i^t) + DT^t + DP_E^t. \]

The energy consumption for transmitting data from the MD to $v_{MD}^t$, $EU^t$, is defined as

\[ EU^t = \frac{x^t \cdot d^t}{u^t \cdot \rho^t} \cdot p_{MD}^t. \]

Let $ED^t$ be the energy consumption for the MD to receive the result data from $v_{MD}^t$ when the offloaded data is completely executed. $ED^t$ is calculated by

\[ ED^t = \frac{\xi_{N^t-1} \cdot x^t \cdot d^t \cdot c^t}{down^t} \cdot p_{MD}^t. \]

where $p_{MD}^t$ is the receiving power of the MD. The energy consumption for edge computing, $EE^t$, is defined as

\[ EE^t = EU^t + ED^t. \]

In addition, this paper considers usage charge when edge computing incurs. If a BS hosts a VNF $f_i^t \in F^t$, the system operator charges the MD because it rents the computing resources at the EI. The usage charge is calculated based on the price per time unit at the computing capacity $cp(f_i^t)$, which is defined as $\Delta(f_i^t) = e^{-\alpha \cdot (e^{cp(f_i^t)} - 1)} \cdot \beta$, where $\alpha$ and $\beta$ are two coefficients [9]. The usage charge for hosting $F_i^t$ at the EI is calculated by

\[ UC^t = \sum_{i=1}^{N^t} \frac{\xi_{i-1} \cdot x^t \cdot d^t \cdot c^t}{cp(f_i^t)} \cdot \Delta(f_i^t). \]

C. Problem Formulation

The POSMJO problem consists of two closely related decision-making steps, namely task partition and VNF placement. The first step decides the amount of task data to be offloaded to the EI. The second step places the corresponding VNFs on BSs to process the offloaded data. The first step is a prerequisite for the second step, and the result of the second step directly affects the performance of the first one. Thus, the two steps are complementary and closely related.

This paper considers three important indicators for measuring the offloading performance, i.e., task execution delay, energy consumption of the MD, and usage charge for edge computing. The execution delay directly determines the task completion time. Energy consumption is important as the MD is with limited battery. Usage charge depends on the budget of the MD for computation offloading service. Hence, the POSMJO problem is multi-objective in nature, with each performance indicator to be minimized. This paper adopts the weighted sum method [44, 45], to aggregate the three performance indicators into a single-objective cost function to minimize, which is defined as

\[ Cost(u^t) = \omega_1 DC^t + \omega_2 EC^t + \omega_3 UC^t, \]

where $\omega_1 + \omega_2 + \omega_3 = 1$, and $DC^t$ is the execution delay of task $u^t$, defined as

\[ DC^t = \max \{ DL^t, DE^t \}. \]

$DC^t$ is equal to $DL^t$ if the local computing delay $DL^t$ is larger than edge computing delay $DE^t$, and $DC^t$ equals $DE^t$, otherwise. $EC^t$ is the energy consumption of the MD executing task $u^t$, which is the summation of the energy consumption incurred by local and edge computing, that is,

\[ EE^t = EL^t + ED^t. \]

The POSMJO problem aims to minimize the average long-term cost, written as,

\[ \min_{x^t, \rho^t} \frac{1}{T} \sum_{t=1}^{T} Cost(u^t), \]

subject to:

\[ C1 : x^t \in [0, 1], \]
\[ C2 : \rho_i^t \in V, i = 1, \ldots, N^t, \]
\[ C3 : DL^t \leq D_{\text{max}}, \]
\[ C4 : cp(f_i^t) \leq cp(f_{MD}, i = 1, \ldots, N^t, \]
\[ C5 : \rho_i^t \leq cp_{MD}^t, i = 1, \ldots, N^t, \]
\[ C6 : bw(f_i^t) = \min \{ bw_e \mid \forall e \in E_{i+1}, i = 1, \ldots, N^t - 1, \]
\[ C7 : DL^t \leq D_{\text{max}}. \]

Constraint $C1$ specifies that an arbitrary amount of the input data of $u^t$ can be offloaded to the EI and executed there. Constraint $C2$ specifies that an arbitrary VNF $f_i^t \in F^t$ can be instantiated on any BS. Constraints $C3$ and $C7$ mean that the local computing delay and edge computing delay are no larger than the execution deadline of $u^t$. Constraints $C4$ and $C5$ guarantee that the computing resources consumed by $f_i^t \in F^t$ cannot exceed the available computing capacity of the MD and the selected BS, respectively. Constraint $C6$ guarantees that for any link $e \in E$, the total bandwidth consumption caused by a data-flow cannot exceed its available bandwidth capacity, where $E_{i+1}$ is the link set of the path between the BSs hosting $f_i^t$ and $f_{i+1}^t$. The POSMJO problem aims to minimize the average long-term cost, written as,

\[ \min_{x^t, \rho^t} \frac{1}{T} \sum_{t=1}^{T} Cost(u^t), \]
We devise a cooperative dual-agent DRL (CDADRL) algorithm to realize the two closely related decision-making steps. The framework of the proposed CDADRL algorithm is depicted in Fig. 2. In CDADRL, the NFV-enabled MEC system consists of two environments, namely the MD and EI environments, respectively. Let \( A_0 \) and \( A_p \) denote the two agents interacting with environments MD and EI, respectively. Based on twin delayed DDPG (TD3), agent \( A_0 \) is in charge of task partition, and it works in the MD. Based on dueling DDQN, \( A_p \) is in charge of VNF placement, and it works in the NFV-MANO node. Agents \( A_0 \) and \( A_p \) can gain their interest information through information exchange process between the MD and EI environments.

### A. MDP Formulation

A DRL-solvable problem is usually modeled as a Markov decision process (MDP). An MDP can be denoted by a 3-tuple \( \langle S, A, r \rangle \), where \( S \) is the state space, and \( A \) is the action space, and \( r(s,a) \) is the reward function, \( s \in S, a \in A \). In time slot \( t \), an agent at state \( s^t \) takes an action \( a^t \) based on a policy \( \varphi \), where \( \varphi \) is a specific state-to-action mapping, \( \varphi : S \rightarrow A \). Then, the environment transits to a new state \( s^{t+1} \) and returns an immediate reward \( r^t \leftarrow r(s^t, a^t) \). The agent aims at learning a policy that maximizes future rewards, through interacting with the environment. The state-action value function corresponding to policy \( \varphi \), \( Q_\varphi(s,a) \), can be described as

\[
Q_\varphi(s,a) = \mathbb{E}_\varphi \left[ \sum_{k=t}^{T} \gamma^{k-t} \cdot r^k \mid s^t = s, a^t = a \right],
\]

where \( \gamma \in [0,1] \) is a discount factor reflecting how important the future rewards are.

#### 1) MDP for Mobile Device Environment:

Given the MD environment, we train \( A_0 \) to find an optimal policy on task partition that minimizes the execution delay, energy consumption and usage charge. The state space, action space, and reward space associated with the MD environment are defined as follows.

1) **State space:**

\[
S_0 = \{ s^t_0 \mid s^t_0 = (F^t, d^t, c^t, D^t_{\text{max}}, cp_{MD}) \}, t = 1, \ldots, T \},
\]

where \( s^t_0 \) is the state of the MD environment in time slot \( t \). \( s^t_0 \) is a vector of the required SFC of \( u^t \), input data size of \( u^t \), number of CPU cycles required to execute \( u^t \), execution deadline of \( u^t \), and available computing capacity of the MD.

2) **Action space:**

\[
A_0 = \{ a^t_0 \mid a^t_0 \in [0,1], t = 1, \ldots, T \},
\]

where \( a^t_0 \) is an action that \( A_0 \) takes in \( s^t_0 \). In time slot \( t \), the MD partitions the input data of task \( u^t \) into local and edge computing parts. We set \( x^t = a^t_0 \) if \( x^t = 0 \), task \( u^t \) is executed on the MD locally. Otherwise, a proportion of the input data of \( u^t \), \( x^t \cdot d^t \),
is executed at the EI, while the rest of the input data is processed on the MD.

3) Reward function: Our objective is to minimize the average long-term cost consisting of the execution delay $DC^t$, energy consumption $EC^t$, and usage charge $UC^t$. So the reward function of the MD environment is set to the negative value of $Cost(u^t)$, defined as

$$r^t_o = \begin{cases} -Cost(u^t), & \text{if constraints } C1 - C7 \text{ are met} \\ -\varrho, & \text{otherwise} \end{cases}, \quad (22)$$

where $Cost(u^t) = \omega_1 DC^t + \omega_2 EC^t + \omega_3 UC^t$, and $\varrho$ is a sufficiently large number. Note that, $EC^t$ and $UC^t$ are obtained after the results are received from the EI environment. $-\varrho$ in reward function (22) is used to tell $A^o$ the importance of choosing actions that satisfy constraints $C1 - C7$.

2) MDP for Edge Infrastructure Environment: Given the EI environment, we train $A^o$ to find an optimal policy on VNF placement that minimizes the edge computing delay and usage charge. As aforementioned, a solution to VNF placement specifies where each VNF in a given SFC is placed. Within time slot $t$, VNF placement of $u^t$ can be divided into $N^t$ stages. At stage $i$, $i = 1, \ldots, N^t$, $A^o$ only decides where to place the $i$-th VNF in $F^t$ at the EI. The state space, action space, and reward space associated with the EI environment are defined as follows:

1) State space:

$$S^o = \{ s^o|^i {\mid s^o|^i = (F^t, d^t, D_{max}^t, v^t_{MD}, a^t, L^i, c_{p^o}^o|i, bw^o|^i, t = 1, \ldots, T) \}, \quad (23)$$

where $s^o|^i$ is the state of the EI environment at stage $i$ of $t$. In state $s^o|^i$, $F^t$, $d^t$, and $D_{max}^t$ are the required SFC, input data size, and execution deadline of $u^t$, respectively. $v^t_{MD}$ is the closest BS to the MD, and $x^t$ is the offloading ratio. $L^i = (\rho_1^t, \ldots, \rho_{N^t}^t)$ is used to identify the BSs selected to host VNFs $f^i_j \in F^t$. Because $\rho_j^t$ cannot be obtained when $A^o$ observes $s^o|^i$, we set $\rho_j^t = -1$. $c_{p^o}^o|^i$ is $\{ c_{p^o}^o|^i \mid \forall v \in V \}$ represents the available computing capacities of all BSs. $bw^o|^i$ is $\{ bw^o|^i \mid \forall e \in E \}$ stands for the available bandwidth capacities of all wired links between BSs.

2) Action space:

$$A^o = \{ a^o|^i | a^o|^i \in V, t = 1, \ldots, T \}, \quad (24)$$

where $a^o|^i$ is an action $A^o$ takes in $s^o|^i$. At each stage, we set $\rho_j^t = a^o|^i$, indicating BS $\rho_j^t$ has been selected to host VNF $f^i_j$. Therefore, we can place all VNFs $f^i_j \in F^t$ after $N^t$ stages.

3) Reward function: Agent $A^o$ is trained to minimize the edge computing delay $DE^t$ and usage charge $UC^t$ in time slot $t$, with a number of constraints corresponding to the VNF placement step satisfied. The reward function of the EI environment is defined as

$$r^t_o = -(R^t_{reward} + R^t_{penalty}), \quad (25)$$

where $R^t_{reward} = \omega_1 DE^t + \omega_2 UC^t$ is the weighted sum of edge computing delay and usage charge, $\omega_1 + \omega_2 = 1$, $R^t_{penalty} = \sum_{j=3}^7 \mu_j \vartheta_j$ is used to punish $A^o$ when its action cannot satisfy constraints $C5 - C7$, where $\mu_j$ is a penalty factor (a sufficiently large number in this paper) and $\vartheta_j \in \{0, 1\}$ is a binary variable. We set $\vartheta_j = 0$ if $C_j$ is satisfied, and $\vartheta_j = 1$, otherwise. Note that, we cannot obtain $r^t_{p,i} = 0$ immediately when $i < N^t$. This is because VNF placement is a process of $N^t$ stages and we cannot obtain $DE^t$ and $UC^t$ before all VNFs $f^i_j \in F^t$ are placed. Thus, we devise a delayed update reward (DUR) scheme to update rewards $r^t_{p,1}, \ldots, r^t_{p,N^t-1}$. To be specific, we set $R^t_{reward} = 0$ when $i < N^t$. After all VNFs are placed, we have $i = N^t$ and we calculate $R^t_{reward}$ by

$$R^t_{reward} = \omega_1 DE^t + \omega_2 UC^t. \quad \text{Then, we update the reward values}$$

$$r^t_{p,i} = r^t_{p,i} - R^t_{reward}, i = 1, \ldots, N^t - 1. \quad (26)$$

As mentioned before, the VNF placement process within $t$ consists of $N^t$ stages. At each stage, a VNF is selected to be placed on one BS. It is possible that the $i$-th VNF cannot be placed successfully, $i \leq N^t$. Simply setting all rewards $r^t_{p,i}, i = 1, \ldots, N^t$, to the same value may cause agent $A^o$ to ignore some reasonable actions in the future. The DUR scheme avoids such situation.

B. TD3-Based Task Partition

As aforementioned, task partition is a continuous optimization problem, i.e., the action space of $A^o$ is continuous. DDPG, a classic DRL algorithm, is widely used to solve problems with continuous action space [16]. However, DDPG often suffers from overestimation of the Q-value and results in sub-optimal policy [46]. TD3 [47], the direct successor of DDPG, overcomes the above problem by leveraging three novel mechanisms, namely the clipped double Q-learning for actor-critic, target policy smoothing, and delayed policy updates.

This paper adapts TD3 to the task partition process. Different from DDPG that learns a single Q-function, TD3 adopts two critic networks and one actor network, as shown in Fig. 2. The actor network consists of two subnets, i.e., one evaluate network $\pi_a$ with parameter $\phi$ and one target network $\pi_{o'}$ with parameter $\phi'$. The critic networks include two evaluate networks $Q_{\theta_1}, Q_{\theta_2}$ with parameters $\theta_1, \theta_2$, and two target networks $Q_{\theta_1}', Q_{\theta_2}'$ with parameters $\theta_1', \theta_2'$.

1) Clipped Double Q-Learning for Actor-Critic: Evaluate networks $Q_{\theta_1}$ and $Q_{\theta_2}$ use a single target $y_o^i$, i.e., the smaller value output by target networks $Q_{\theta_2}'$ and $Q_{\theta_2}'$, defined as

$$y_o^i = r_o^i + \gamma \min_{a_o^i \in A_o} Q_{\theta_1}'(s_o^{i+1}, a_o^i), \quad (27)$$

where $a_o^i$ is the action generated by the target policy smoothing scheme, as defined in (29). We update the parameters $\theta_i$ of $Q_{\theta_i}$, $i = 1, 2$, by minimizing the loss function, defined as

$$L(\theta_i) = \frac{1}{M} \sum_{j=1}^M (y_o^j - Q_{\theta_i}(s_o^j, a_o^j))^2, i = 1, 2, \quad (28)$$

where $M$ is the number of transitions in a mini-batch.

2) Target Policy Smoothing: The action, $a_o^i$, used to calculate target $y_o^i$ is obtained from the target action network $\pi_{o'}$, with clipped noise considered. After adding the noise, we also clip $a_o^i$ to lie in the valid action range $[a_o^{low}, a_o^{high}]$, ensuring $a_o^i \in A_o$. Therefore, $a_o^i$ is defined as

$$a_o^i = \text{clip} \left( \pi_{o'}(s_o^{i+1}) + \text{clip}(\epsilon, -c, c), a_o^{low}, a_o^{high} \right), \quad (29)$$

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
Algorithm 1: TD3-Based Task Partition.

1: Initialize evaluate critic networks $Q_{\theta_1}$ and $Q_{\theta_2}$, and evaluate actor network $\pi_{\phi}$ with random parameters $\theta_1$, $\theta_2$, and $\phi$;
2: Initialize target networks $Q_{\theta_1}'$, $Q_{\theta_2}'$, $\pi_{\phi'}$ with $\theta_1' \leftarrow \theta_1$, $\theta_2' \leftarrow \theta_2$, $\phi' \leftarrow \phi$;
3: Set replay buffer $B_0$ to $\emptyset$;
4: for $k = 1, \ldots, EPS_{\text{max}}$ do
5: Observe state $s_1$;
6: for $t = 1, \ldots, T$ do
7: Select action $a_o^t = \text{clip}(\pi_{\phi}(s_o^t)) + \epsilon, 0, 1)$;
8: Take action $a_o^t$ and calculate local computing results $DL^t$ and $EL^t$;
9: Send offloading ratio $x^t = a_o^t$ to the EI environment;
10: $A_o$ places all VNFs in $F^t$ on BSs for edge computing purpose;
11: Receive edge computing results $DE^t$, $EE^t$ and $UC^t$ from the EI environment;
12: Based on the local and edge computing results, calculate $r_t^1$ and observe $s_o^{t+1}$;
13: Store transition $(s_o^t, a_o^t, r_t^1, s_o^{t+1})$ in $B_o$;
14: Sample a mini-batch of $M$ transitions $(s_o^j, a_o^j, r_o^j, s_o^{j+1}), j = 1, \ldots, M$, from $B_o$;
15: Calculate $\tilde{a}_o$ according to (29);
16: Set $y^t = r^t + \gamma \min_{a_o^t} Q_{\theta_1}(s_o^{t+1}, a_o^t)$;
17: Update evaluate critic networks $\theta_i \leftarrow \min_{\theta_i} \frac{1}{M} \sum_{j=1}^{M} (y^j - Q_{\theta_i}(s_o^j, a_o^j))^2$, $i = 1, 2$;
18: if $(t \mod K) = 0$ then
19: Update $\phi$ by the deterministic policy gradient:
20: Update target networks with $\theta_i' \leftarrow \tau \theta_i + (1 - \tau) \theta_i'$, $i = 1, 2$;
21: end if
22: end for
23: end for

where $\epsilon \sim \mathcal{N}(0, \sigma)$ is the random noise defined by parameter $\sigma$, and $\epsilon > 0$. According to the action space in (21), we have $a_o^{\text{low}} = 0$ and $a_o^{\text{high}} = 1$. Here, the noise $\epsilon$ is a regularizer for TD3, helping it to achieve smoother state-action value estimation.

3) Delayed Policy Updates: TD3 updates evaluate actor network $\pi_{\phi}$ and target networks $Q_{\theta_1}'$, $Q_{\theta_2}'$ and $\pi_{\phi'}$ once, after the evaluate critic networks $Q_{\theta_1}$ and $Q_{\theta_2}$ are updated $K$ times, which is referred to as delayed policy updates [47]. That is, policy and target networks are not updated unless $Q_{\theta_1}$ and $Q_{\theta_2}$ are updated sufficiently. We use parameter $\theta_1$ of $Q_{\theta_1}$ to update $\pi_{\phi}$ via the policy gradient method as

$$\nabla_{\phi} J(\phi) = \frac{1}{M} \sum_{j} \nabla_{a_o^j} Q_{\theta_1}(s_o^j, a_o^j) |_{a_o = \pi_{\phi}(s_o^j)} \nabla_{\phi} \pi_{\phi}(s_o^j),$$

(30)

After that, we adopt soft update to gradually update $\theta_1'$, $\theta_2'$, and $\phi'$. Soft update is a method that updates the network by combining the parameters of evaluate and target networks, i.e., the parameters of target networks are updated by having them gradually track the evaluate networks, as defined in (31).

$$\theta_i' \leftarrow \tau \theta_i + (1 - \tau) \theta_i', \ i = 1, 2,$$

$$\phi' \leftarrow \tau \phi + (1 - \tau) \phi',$$

(31)

where $\tau \ll 1$ is the learning rate. By using soft update, the target values are constrained to change gradually, which greatly improves the stability of learning [16].

The pseudo-code of TD3-based task partition is shown in Algorithm 1. At the beginning, agent $A_o$ initializes the evaluate and target networks. The replay buffer of $A_o$, $B_o$, is set to $\emptyset$. Then, $A_o$ starts interacting with the MD environment. In time slot $t$, $A_o$ generates an action according to the current state, defined as

$$a_o^t = \text{clip}(\pi_{\phi}(s_o^t)) + \epsilon, 0, 1),$$

where $\epsilon \sim \mathcal{N}(0, \sigma)$ is the exploration noise defined by parameter $\sigma$. As shown in Fig. 2, right after action $a_o^t$ is generated, the MD environment sends offloading ratio $x^t = a_o^t$ to the EI environment and receives edge computing results $DE^t$, $EE^t$ and $UC^t$. Then, the MD environment transits to the next state $s_o^{t+1}$ and feeds back reward $r_o^t$ to $A_o$. Transition $(s_o^t, a_o^t, r_o^t, s_o^{t+1})$ is added to the replay buffer $B_o$.

In the training process, we randomly sample a mini-batch of $M$ transitions from $B_o$. Based on target network $\pi_{\phi'}$ and state $s_o^{t+1}$, we generate action $\tilde{a}_o$ in favor of target policy smoothing. We use (27) to calculate target value $y_o^t$. After that, we update the two evaluate networks of critic $\theta_1$ and $\theta_2$ by minimizing the loss function defined in (28). The deterministic policy gradient method in (30) is adopted to update the evaluate actor network.

Then, we soft update target networks $\theta_1'$, $\theta_2'$, and $\phi'$ according to (31). Finally, after training $A_o$ $EPS_{\text{max}}$ rounds, we get the optimal policy for task partition.

C. Dueling DDQN-Based VNF Placement

VNF placement is a discrete optimization problem, i.e., the action space of $A_p$ is non-continuous. Model-free DRL approaches like DQN and DDQN are value-based algorithms [48], which have been widely applied to tackle such problems. For most DRL tasks, the values of a Q-function are distinct in different state-action pairs. Nevertheless, in some states, actions have no significant impact on the Q-function values. To solve this problem, dueling network architecture has been proposed in [49]. The dueling network separates the representation of state values and action advantages, which can learn state values without influence of actions. Dueling DDQN proposed in [15] combines the advantages of dueling network architecture and DDQN, making it suitable for the POSMJO problem formulated in this paper. Therefore, dueling DDQN is adopted to the VNF placement process.

As shown in Fig. 2, the dueling DDQN consists of an evaluate network $Q_{\theta_1}$ with parameter $\theta_1$, and a target network $Q_{\theta_1}'$ with parameter $\theta_1'$. Dueling network architecture is adopted in both
\[ Q_{\theta_p} \text{ and } Q_{\theta_o}. \] For example, the evaluate network, \( Q_{\theta_p} \), consists of two streams that represent the value function \( V_{\theta_p} \) and advantage function \( A_{\theta_p} \), defined as
\[
Q_{\theta_p}(s^p_t, a^p_t) = V_{\theta_p}(s^p_t; \beta) + A_{\theta_p}(s^p_t, a^p_t; \alpha),
\]
where \( \beta \) and \( \alpha \) are the parameters of \( V_{\theta_p} \) and \( A_{\theta_p} \), respectively. \( V_{\theta_p} \) is only related to states, while \( A_{\theta_p} \) is related to both states and actions. Eq (32) can be directly applied to get the state-action value. However, this equation is unidentifiable, e.g., given \( Q_{\theta_p} \), we cannot recover \( V_{\theta_p} \) and \( A_{\theta_p} \) uniquely [49]. In order to obtain identifiability, \( Q_{\theta_p} \) is defined as
\[
Q_{\theta_p}(s^p_t, a^p_t) = V_{\theta_p}(s^p_t; \beta) + \left( A_{\theta_p}(s^p_t, a^p_t; \alpha) - \frac{1}{|A_p|} \sum_{a_p \in A_p} A_{\theta_p}(s^p_t, a_p; \alpha) \right),
\]
where \( |A_p| \) is the cardinality of \( A_p \).

In dueling DDQN, the loss function reflecting the estimation performance, is defined as
\[
L(\theta_p) = \mathbb{E}_{s^p_t, a^p_t, r^p_t, s^{p+1}_t} \left[ (y^p_t - Q_{\theta_p}(s^p_t, a^p_t))^2 \right],
\]
where \( y^p_t \) is the target defined as
\[
y^p_t = r^p_t + \gamma Q_{\phi_p}(s^{p+1}_t, \arg \max_{a_p} Q_{\theta_p}(s^{p+1}_t, a_p)).
\]

The loss function in (34) can be minimized by stochastic gradient descent for updating parameter \( \theta_p \) [48].

The pseudo-code of dueling DDQN-based VNF placement is shown in Algorithm 2. At the beginning, agent \( A_p \) initializes the evaluate and target networks, respectively. Different from \( A_o \), \( A_p \) is associated with two replay buffers, i.e., \( B_p \) and \( B'_p \). \( B_p \) is used to store transitions during training, while \( B'_p \) is adopted to store transitions corresponding to the VNF placement decisions of \( u^t \) in time slot \( t \). After \( Q_{\theta_p}, Q_{\phi_p} \), and \( B_p \) are initialized, \( A_p \) starts interacting with the EI environment.

Recall that, within \( t \), the VNF placement of \( u^t \) is divided into \( N^t \) stages. Within \( t \), \( A_p \) receives the offloading ratio, \( x^t \), from the MD environment. Then, at stage \( i \), \( A_p \) decides where to place the \( i \)-th VNF in \( F^i \), \( i = 1, \ldots, N^t \). Due to lack of experience in the early process, \( A_p \) generates an action by an \( \varepsilon \)-greedy strategy, defined as
\[
a_{p,i}^{t,i} = \begin{cases} 
\text{a random action}, & \text{with probability } \varepsilon \\
\arg \max_{a_p} Q_{\theta_p}(s_{p,i}^t, a_p), & \text{with probability } 1 - \varepsilon
\end{cases}
\]

Subsequently, the EI environment transits to the next state \( s_{p,i}^{t,i+1} \) and feeds back reward \( r_{p,i}^{t,i} \) to \( A_p \). Transition \( (s_{p,i}^{t,i}, a_{p,i}^{t,i}, r_{p,i}^{t,i}, s_{p,i}^{t,i+1}) \) is added to the replay buffer \( B'_p \). After all VNFs in \( F^1 \) are placed, the DUR scheme is adopted to update the rewards of transitions in \( B'_p \) according to (26). After DUR, all transitions in \( B'_p \) are stored in \( B_p \). Then, edge computing results, \( DE^t, EE^t \), and \( UC^t \), are sent to the MD environment.

In the training process, we randomly sample a mini-batch of \( M \) transitions from \( B_p \). We calculate the target value \( y_{p,t}^i \) according to (35). Gradient descent is performed, with the loss function defined in (34). After that, we update the target network parameter \( \theta'_p \) every \( K \) steps. Finally, after training \( A_p \), \( EPS_{\max} \) rounds, we get the optimal policy for VNF placement.

\section{Cooperative Dual-Agent DRL Algorithm}

In the CDADRL algorithm, agents \( A_o \) and \( A_p \) collaborate on tackling the POSMIO problem. The state, \( s_{p,i}^t \) of \( A_p \), depends on the offloading ratio \( x^t = a_o^t \) made by \( A_o \) and the reward, \( r_{o,i}^t \) of \( A_o \), is calculated based on the edge computing results corresponding to the VNF placement solution made by \( A_p \). As shown in Fig. 2, the collaboration between the two agents is realized by the information exchange between the MD and EI environments. The swimlane diagram shown in Fig. 4 depicts the interaction process between the agents and environments. To be specific, between \( A_o \) and the MD environment, \( A_o \) observes the current state \( s_{o,i}^t \) and takes action \( a_o^t \). After receiving the edge computing results from the EI environment, \( DE^t, EE^t \) and \( UC^t \), the MD environment feeds back reward \( r_{o,i}^t \) to \( A_o \) and transits to the next state \( s_{o,i}^{t+1} \). Similarly, after the MD
The flowchart of the proposed CDADRL algorithm is shown in Fig. 3. It can be seen that, \( \mathbf{A}_o \) and \( \mathbf{A}_p \) interact with each other via the information exchange process between two environments. The two agents realize the two related decision-making steps through close cooperation. At the beginning, \( \mathbf{A}_o \) and \( \mathbf{A}_p \) initialize their neural networks and replay buffers respectively. In time slot \( t \), \( \mathbf{A}_o \) decides the offloading ratio \( x_t \) first. After receiving \( x_t \), \( \mathbf{A}_p \) takes a series of actions to generate a VNF placement solution and stores transitions to \( B_p \). Then, \( \mathbf{A}_o \) receives edge computing results and stores transition in \( B_o \). After that, \( \mathbf{A}_o \) and \( \mathbf{A}_p \) update their neural networks respectively. After training \( \mathbf{A}_o \) and \( \mathbf{A}_p \) \( \text{EPS}_{\text{max}} \) rounds, we can get the optimal policies for task partition and VNF placement.

V. SIMULATION RESULTS AND DISCUSSION

In this section, we evaluate the performance of the proposed CDADRL algorithm for the POSMJO problem. A python simulator running on Google TensorFlow 2.2 is developed for performance evaluation. In the simulation, we assume that the number of time slots is 20, i.e., \( T = 20 \), and the duration of each time slot is 5 seconds. Following existing works, we assume an edge infrastructure comprising 10 BSs, each co-located with an MEC server, providing edge computing services to the MD. The minimum distance between any two neighboring BSs is 1600 m, as reported in [9]. The MD receives computation offloading services by communicating with the closest BS to it, according to [35]. The distance between the MD and its closest BS is randomly generated in the range of [100, 800] m [9]. The computing capacity of each BS, \( cp_v \), is randomly generated in the range of [2, 10].
TABLE III
EXPERIMENTAL PARAMETER SETUP

| Parameter                          | Value      |
|-----------------------------------|------------|
| Number of BSSs                    | 10         |
| Wireless bandwidth (W)            | 20 MHz     |
| Noise power (ρ^n)                 | 10^{-6} Watt |
| Computing capacity of the MD (ρ_{MD}) | 0.6 GHz   |
| Transmission power of the MD (ρ_{MD}') | 0.5 Watt |
| Receiving power of the MD (ρ_{MD}^r) | 0.1 Watt |
| Transmission power of each BS (ρ_{BS}) | [1, 2] Watt |
| Computing capacity of each BS (ρ_{BS}) | [2, 6] GHz |
| Bandwidth capacity of each link (bw_{e}) | [20, 100] Mbps |
| Input data size of each task (d)  | [800, 1000] Kbits |
| Number of CPU cycles (c)          | [800, 1000] cycle/bit |
| Number of VNFs in an SFC          | [3, 5]     |
| Required computing capacity of each VNF (ρ_{i,j}) | [0.1, 0.5] GHz |
| Required bandwidth between any two VNFs (bw_{i,j}) | [5, 10] Mbps |
| Ratio of h_{i} to d' (ξ_{i})      | [0.5, 1.5] |
| Execution deadline (D_{max})       | [20, 35] Sec |
| Effective capacitance coefficient (κ) | 10^{-26} |
| Discount factor (γ)               | 0.99       |
| Maximum number of episodes (E_{P_{max}}) | 3000     |

A. Parameter Study of CDADRL Algorithm

We study the impact of parameters on the performance of the proposed CDADRL algorithm, including the learning rate, batch size, and buffer size. The average cumulative reward of CDADRL algorithm is depicted in Fig. 5. Fig. 5(a) shows the cumulative rewards with different learning rates. As can be seen, both 0.0001 and 0.01 lead to poor training performance, and the cumulative reward curve oscillates at low values. This is because a too large learning rate leads to rapid changes to the parameters of neural networks, which makes the training process unstable. A too small learning rate results in trivial changes to the neural networks’ parameters, which leads to painfully slow convergence. Obviously, 0.001 is more appropriate than 0.01 and 0.0001. Thus, we hereafter fix the learning rate to 0.001. Fig. 5(b) shows the cumulative rewards with different batch sizes. With a small batch size, i.e., 64, the algorithm cannot reach to high cumulative reward values because the gradients from a small batch may not cover promising directions of optimization. On the contrary, a too large batch size results in frequent use of non-effective experiences. Therefore, we hereafter set the batch size to 128. Fig. 5(c) depicts the cumulative rewards with different buffer sizes. Clearly, the algorithm with a buffer size of 2000 obtains the best performance on training. This is because a smaller buffer stores less experiences for policy optimization and it is less likely to keep promising ones. A larger buffer maintains more promising experiences. However, out-of-date transitions are more likely to be picked up, disturbing the current direction of optimization. Hence, we hereafter set the buffer size to 2000.

B. Overall Performance Evaluation

Note that the POSMJO problem is brand-new and no existing algorithm is immediately available to tackle it. In order to evaluate the performance of the CDADRL algorithm, we compare it with three combinations of state-of-the-art DRL algorithms for continuous and discrete decision-making, against cumulative reward. These combinations are TD3-DDQN, DDPG-Duel, and DDPG-DDQN, listed below.

- **TD3-DDQN**: the original CDADRL algorithm with A_p based on the conventional DDQN [48] instead of dueling DDQN.
- **DDPG-Duel**: the original CDADRL algorithm with $A_\alpha$ based on DDPG [16] instead of TD3.
- **DDPG-DDQN**: the original CDADRL algorithm with $A_\alpha$ based on DDPG and $A_p$ based on DDQN.
- **CDADRL**: The cooperative dual-agent DRL algorithm proposed in this paper.

Fig. 6 shows the cumulative reward curves of the four algorithms with different computing capacities of the MD. It is easily seen that **CDADRL** and **TD3-DDQN** perform the best and second-best among the four algorithms. Compared to **TD3-DDQN**, **CDADRL** has a faster training speed and higher cumulative reward values once converged. This is because Dueling DDQN helps CDADRL to make better use of the state value, not just the state-action value, which is in favor of making more appropriate VNF placement decisions. Besides, **DDPG-Duel** and **DDPG-DDQN** are the two worst algorithms as they have poor performance on convergence. For example, in Fig. 6(c) the **DDPG-Duel** algorithm does not well converge after 3000 episod. The reason behind this is the task partition step of **DDPG-Duel** and **DDPG-DDQN** is based on DDPG. DDPG is prone to overestimating Q-values, making it easily trapped into sub-optimal policies [46]. On the other hand, **CDADRL** adopts TD3 which leverages three novel mechanisms to overcome the above difficulty. Thus, the combination of TD3 and Dueling DDQN is the most appropriate for addressing the POSMJO problem.

We also evaluate the offloading performance by comparing **CDADRL** with four baseline offloading schemes and two combinatorial optimization-based algorithms, in terms of execution delay, energy consumption of the MD, and usage charge for edge computing. The six algorithms are listed below.

- **Local**: Any incoming task is executed on the MD, which means no task is to be offloaded to the edge.
- **Edge**: All incoming tasks are offloaded to the MEC servers for processing, where the DQN-based online SFC deployment method is adopted to place VNFs on MEC servers [27].
- **Binary**: An incoming task is either executed locally on the MD or offloaded completely to the edge for processing. Assume a task is executed locally with a probability of 50%. The online algorithm proposed in [29] is applied to VNF placement.

- **Random**: For an arbitrary incoming task, its offloading ratio is set to a number randomly generated in the range of [0, 1]. The algorithm randomly selects a BS in $V$ to place the $i$-th VNF in $F^1$ until all VNFs in $F^4$ are placed.
- **PSO-GA**: This algorithm employs particle swarm optimization (PSO) [53] and genetic algorithm (GA) [54] for addressing the task partition and VNF placement subproblems, respectively. The population size and the number of iterations are set to 50 and 200, respectively. For PSO, the cognitive acceleration parameter and social acceleration parameter are both set to 0.5. For GA, the crossover and mutation rates are set to 0.5 and 0.2, respectively.
- **GWO-STS**: This algorithm uses grey wolf optimization (GWO) [55] for task partition and adopts the STS heuristic algorithm proposed in [29] for VNF placement. For GWO, the population size and the number of iterations are set to 20 and 1000, respectively; all the acceleration coefficients are set to 0.5; the mutation possibility is set to 0.04.

Fig. 7 depicts the results of average execution delay (AED), average energy consumption (AEC) and average usage charge (AUC) of the seven algorithms with different computing capacities of the MD. It is observed that in most cases the AED and AUC values decrease, and AEC value increases as the MD’s computing capacity increases. Obviously, **CDADRL** achieves the best AED value in each $c_{PM, D}$ case. This is because CDADRL makes better decisions on partial offloading and VNF placement simultaneously, compared with the other six algorithms. Both **PSO-GA** and **GWO-STS** optimize task partition and VNF placement as two individual parts separately, resulting the second and third best AED values. As shown in Fig. 7(b), **Edge** and **Local** obtain the best and worst AEC values, respectively. In **Edge**, whenever a task arrives, it is offloaded to the MEC servers for processing. In this case, the energy consumption incurred on the MD only involves transmitting the task data to the MEC servers and receiving the processed task from them. As for **Local**, any incoming task will be processed on the MD, leading to the highest energy consumption. Besides, it is clearly seen that **CDADRL** outperforms **Binary**, **Random**, **PSO-GA** and **GWO-STS** regarding AEC. As can be seen in Fig. 7(c), **Edge** and **Local** obtain the worst and best AUC results, respectively. **Local** always costs $0$ for edge computing in all test scenarios, as expected. **Edge** offloads all required tasks to

![Fig. 6. Cumulative reward of the four algorithms with different computing capacities of the MD.](image-url)
the MEC servers, consuming more computing resources than the other algorithms. This is the rationale behind the highest AUC values the other algorithms leads to. On the other hand, CDADRL achieves better performance on compromising AED and AUC, than Local, Edge, Binary, Random, PSO-GA and GWO-STS in all test scenarios.

Fig. 8 depicts the AED, AEC and AUC values of the seven algorithms with different task execution deadlines. It is observed that the AED values of CDADRL, PSO-GA and GWO-STS grow slowly when the execution deadline increases. In addition, CDADRL achieves the lowest AED value while achieving the second lowest AEC and AUC values.

Fig. 9 shows the AED, AEC and AUC values of the seven algorithms with different input data sizes. As the input data size grows for each task, the AED, AEC and AUC values gradually increase. Moreover, CDADRL outperforms the other six algorithms in AED and is the second best in terms of AEC and AUC.
Fig. 10 illustrates the AED, AEC, and AUC values of the seven algorithms with different computing capacities of BS.

It is observed that in most cases, the AED and AUC values decrease and the AEC value increases as the BS’s computing capacity increases. In addition, CDADRL obtains the best AED value in each BS’s computing capacity. In both AEC and AUC, CDADRL presents the second best in all cases.

Fig. 11 illustrates the impact of the distance between the MD and its closest BS on the AED, AEC, and AUC values obtained by seven different algorithms. As the distance increases, there is a significant increase in the AED and AEC values and a dramatical decrease in the AUC value. This indicates that as the MD moves away from its closest BS, there is a higher likelihood of utilizing local computation rather than remote execution.

The average computational time (ACT) is considered as the realtime metric that reflects how long in average an algorithm responds to an offloading request of the MD. Fig. 12 depicts the ACT values obtained by the seven algorithms. Clearly, CDADRL returns a solution much faster than PSO-GA and GWOSTS which are combinatorial optimization-based. CDADRL makes decisions for task partition and VNF placement in about 0.1 seconds, meeting the demand for realtime solutions in dynamic NEV-enabled MEC systems.

![Fig. 10](image1.png)

![Fig. 11](image2.png)

![Fig. 12](image3.png)

With the AED, AEC, AUC, and ACT results considered, one can easily conclude that the proposed CDADRL achieves the best offloading performance among the seven baseline algorithms. Obviously, CDADRL is one of the most appropriate algorithms for solving the POSMJO problem.
VI. CONCLUSION AND FUTURE WORK

This paper formulates a partial offloading and SFC mapping joint optimization (POSMJO) problem in the context of NFV-enabled MEC and presents a cooperative dual-agent DRL (CDADRL) algorithm to tackle it. Effective collaboration between the two agents is achieved through information exchange between the MD and EI environments. The TD3-based agent is able to provide high-quality task partition while the duel DDQN-based agent offers appropriate placement for service function chain requests. Simulation results demonstrate the superiority of the proposed CDADRL over three combinations of state-of-the-art DRL algorithms with respect to cumulative reward. Besides, CDADRL significantly outperforms a number of baseline algorithms in terms of execution delay, energy consumption, and usage charge.

In the future, we plan to adapt the proposed CDADRL algorithm to large-scale networks with multiple MDs, where each MD requests the partial offloading and SFC mapping services. In this paper, the execution delay, energy consumption, and usage charge are aggregated into one objective function to minimize. We assume the weights per objective are unchanged during learning and execution. However, in some cases, the weights are not known beforehand and may change over time. Single-objective RLs, e.g., DDPG, DQN, and CDADRL, are not suitable for these scenarios. Thus, we hope to further extend our algorithm to a multi-objective RL method. Besides, we will consider an NFV-enabled MEC scenario with multiple MDs which require computation offloading services while moving. In such a scenario, we aim to develop a competitive and cooperative multi-agent game-based decision-making system, where MDs compete and share the resources of edge servers. Additionally, we will design a task queue at the EI to process VNF placement requests from multiple MDs in a sequential manner.

ACKNOWLEDGMENT

The authors would like to thank the anonymous referees and the associate editor for their valuable comments.

REFERENCES

[1] M. Goudarzi, H. Wu, M. Palaniswami, and R. Buyya, “An application placement technique for concurrent IoT applications in edge and fog computing environments,” IEEE Trans. Mobile Comput., vol. 20, no. 4, pp. 1298–1311, Apr. 2021.

[2] B. Liang, M. A. Gregory, and S. Li, “Multi-access edge computing fundamentals, services, enablers and challenges: A complete survey,” J. Netw. Comput. Appl., vol. 199, 2022, Art. no. 103308.

[3] Y. Sritwirdhana, P. Porambage, M. Liyanage, and M. Yliantila, “A survey on mobile augmented reality with 5G mobile edge computing: A multi-task learning approach,” IEEE Trans. Mobile Comput., vol. 20, no. 9, pp. 2745–2762, Sep. 2021.

[4] F. Song, H. Xing, X. Wang, S. Luo, P. Dai, and K. Li, “Offloading dependent tasks in multi-access edge computing: A multi-objective reinforcement learning approach,” Future Gener. Comput. Syst., vol. 128, pp. 333–348, 2022.

[5] H. Lin, S. Zeadally, Z. Chen, H. Labied, and L. Wang, “A survey on computation offloading modeling for edge computing,” J. Netw. Comput. Appl., 2020, Art. no. 102781.
F. Zhang, G. Han, L. Liu, M. Martinez-Garcia, and Y. Peng, “Deep reinforcement learning based cooperative partial task offloading and resource allocation for IoT applications,” IEEE Trans. Netw. Sci. Eng., to be published, doi: 10.1109/TNSE.2022.3167949.

P. Sun, J. Lan, J. Li, Z. Guo, and Y. Hu, “Combining deep reinforcement learning with graph neural networks for optimal VNF placement,” IEEE Commun. Lett., vol. 25, no. 1, pp. 176–180, Jan. 2021.

J. Bi, H. Yuan, K. Zhang, and M. Zhou, “Energy-minimized partial computation offloading for delay-sensitive applications in heterogeneous edge networks,” IEEE Trans. Emerg. Top. Comput., vol. 10, no. 4, pp. 1941–1954, Fourth Quarter 2022.

Fuhong Song received the MEng degree in computer technology and the PhD degree in computer science and technology from Southwest Jiaotong University, Chengdu, China, in 2018 and 2022, respectively. He is currently a lecturer with the School of Information, Guizhou University of Finance and Economics. His research interests include edge computing, multi-objective optimization, and reinforcement learning.

Xinhan Wang received the BEng degree in computer science and technology from the Southwest University, Chongqing, China, in 2016. He is currently working toward the PhD degree with the School of Computing and Artificial Intelligence, Southwest Jiaotong University, Chengdu, China. His research interests include deep reinforcement learning, mobile edge computing, and network function virtualization.

Huanlai Xing (Member, IEEE) received the BEng degree in communications engineering from Southwest Jiaotong University, Chengdu, China, in 2006, the MSc degree in electromagnetic fields and wave-length technology from the Beijing University of Posts and Telecommunications, Beijing, China, in 2009, and the PhD degree in computer science from the University of Nottingham, Nottingham, U.K., in 2013. He is an associate professor with the School of Computing and Artificial Intelligence, Southwest Jiaotong University. His research interests include evolutionary computation, cloud computing, multi-objective optimization, network function virtualization and software defined networks. He has authored and co-authored more than 80 peer-reviewed journal and conference papers. He is a member of ACM.

Shouxiao Luo (Member, IEEE) received the bachelor’s degree in communication engineering and the PhD degree in communication and information systems from the University of Electronic Science and Technology of China, Chengdu, China, in 2018 and 2022, respectively. He is currently a lecturer with the School of Informatics, Guizhou University of Finance and Economics. His research interests include edge computing, multi-objective optimization, and reinforcement learning.
Penglin Dai (Member, IEEE) received the BS degree in mathematics and applied mathematics and the PhD degree in computer science from Chongqing University, Chongqing, China, in 2012 and 2017, respectively. He is currently an assistant professor with the School of Computing and Artificial Intelligence, Southwest Jiaotong University, Chengdu, China. His research interests include intelligent transportation systems and vehicular cyber-physical systems.

Bowen Zhao received the BEng degree in computer science and technology from Southwest Jiaotong University, Chengdu, China, in 2020. He is currently working toward the PhD degree with the School of Computing and Artificial Intelligence, Southwest Jiaotong University, Chengdu, China. His research interests include deep reinforcement learning, time series classification, and mobile edge computing.