When Deep Reinforcement Learning Meets Federated Learning: Intelligent Multitimescale Resource Management for Multiaccess Edge Computing in 5G Ultradense Network

Shuai Yu, *Member, IEEE*, Xu Chen, *Senior Member, IEEE*, Zhi Zhou, *Member, IEEE*, Xiaowen Gong, *Member, IEEE*, and Di Wu, *Senior Member, IEEE*

***Abstract*—Recently, smart cities, healthcare system, and smart vehicles have raised challenges on the capability and connectivity of state-of-the-art Internet-of-Things (IoT) devices, especially for the devices in hotspots area. Multiaccess edge computing (MEC) can enhance the ability of emerging resource-intensive IoT applications and has attracted much attention. However, due to the time-varying network environments, as well as the heterogeneous resources of network devices, it is hard to achieve stable, reliable, and real-time interactions between edge devices and their serving edge servers, especially in the 5G ultradense network (UDN) scenarios. Ultradense edge computing (UDEC) has the potential to fill this gap, especially in the 5G era, but it still faces challenges in its current solutions, such as the lack of: 1) efficient utilization of multiple 5G resources (e.g., computation, communication, storage, and service resources); 2) low overhead offloading decision making and resource allocation strategies; and 3) privacy and security protection schemes. Thus, we first propose an intelligent UDEC (I-UDEC) framework, which integrates blockchain and artificial intelligence (AI) into 5G UDEC networks. Then, in order to achieve real-time and low overhead computation offloading decisions and resource allocation strategies, we design a novel two-timescale deep reinforcement learning (2Ts-DRL) approach, consisting of a fast-timescale and a slow-timescale learning process, respectively. The primary objective is to minimize the total offloading delay and network resource usage by jointly optimizing computation offloading, resource allocation, and service caching placement. We also leverage federated learning (FL) to train the 2Ts-DRL model in a distributed manner, aiming to protect the edge devices' data privacy. Simulation results corroborate the effectiveness of both the 2Ts-DRL and FL in the I-UDEC framework and prove that our proposed algorithm can reduce task execution time up to 31.87%.**

***Index Terms*—Blockchain, computation offloading, deep reinforcement learning (DRL), federated learning (FL), multiaccess edge computing (MEC), service caching, ultradense network (UDN).**

I. Introduction

NOWADAYS, the advances in cloud computing and communication technology are spawning a variety of intelligent mobile and Internet of Things (IoT) devices, such as smartphones, Google Glass, and smart meters. Accompanied by the intelligent devices, multiple resource-intensive mobile and IoT applications are designed to enhance the user experience, such as wearable cognitive assistance, augmented reality, and collaborative 3-D gaming. Such applications use complex algorithms for camera tracking and object recognition, require extensive computation power, high-speed connection, real-time feedback, and a variety of cloud computing services. Furthermore, the number of such mobile and IoT devices is anticipated to have an explosive growth in the 5G era, which will result in the congestion of the cloud computing network. In light of this, multiaccess edge computing (MEC) [1], [2] and ultradense network (UDN) [3] are expected as effective solutions to meet the computation and communication requirements, respectively. MEC provides a capillary distribution of cloud computing capabilities to the edge of the network, enabling rich services and applications in close proximity to edge devices. UDN can offer adequate baseband resources by deploying more base stations close to the edge users.

Recently, a new research paradigm named ultradense edge computing (UDEC) [4] emerges. It integrates MEC with UDN by deploying MEC servers on the base stations of UDN [5], [6]. This integration not only achieves lower data transmission delay but also meets the requirement of real-time computing. However, how to effectively integrate MEC with UDN becomes a critical problem in the future development of 5G, and is underexplored to the best of our knowledge. Note

Manuscript received April 30, 2020; revised July 22, 2020 and August 8, 2020; accepted September 20, 2020. Date of publication September 24, 2020; date of current version February 4, 2021. This work was supported in part by the National Key Research and Development Program of China under Grant 2017YFB1001703; in part by the National Science Foundation of China under Grant 62002397 and Grant 61972432; in part by the Program for Guangdong Introducing Innovative and Entrepreneurial Teams under Grant 2017ZT07X355; in part by the Pearl River Talent Recruitment Program under Grant 2017GC010465; in part by the Science and Technology Program of Guangzhou under Grant 202007040006; and in part by the Guangdong Special Support Program under Grant 2017TX04X148. *(Corresponding author: Xu Chen.)*

Shuai Yu, Xu Chen, Zhi Zhou, and Di Wu are with the School of Data and Computer Science, Sun Yat-sen University, Guangzhou 510006, China (e-mail: yushuai@mail.sysu.edu.cn; chenxu35@mail.sysu.edu.cn; zhouzhi9@mail.sysu.edu.cn; wudi27@mail.sysu.edu.cn).

Xiaowen Gong is with the Department of Electrical and Computer Engineering, Auburn University, Auburn, AL 36849 USA (e-mail: xgong@auburn.edu).

Digital Object Identifier 10.1109/JIOT.2020.3026589

that an effective integration should consider the following key issues.

1) *Full Use of System Resources:* By leveraging computation offloading [4], a mobile application can be split into local parts (usually communication intensive) and offloaded parts (usually computation intensive). Then, an edge server allocates a variety of system resources (e.g., computation, communication, storage, and service) to process the offloaded parts [7]. The problem becomes extremely complex in the UDEC environments, since UDEC has massive edge devices and edge servers with heterogeneous computation resources (e.g., CPU and GPU) and communication links (e.g., cellular and D2D). Thus, it is necessary to fully and efficiently utilize the system resources to improve the UDEC performance.
2) *Dynamic and Low-Cost Scheduling:* The system resources in UDEC are usually diverse (e.g., various service caching placement strategies, different CPU frequencies of mobile devices and cloud/edge servers, and memory) and time-varying (e.g., network disconnection, channel state, and backhaul latency). The distributed and heterogeneous natures of the resources make the scheduling process more complex. Moreover, computation offloading requires real-time application partitioning and low overhead resource allocation. Thus, dynamic and real-time scheduling schemes are required to enhance the Quality of Service (QoS) of edge devices.
3) *Privacy and security-preserving services* are crucial to the MEC-enabled UDN, since the interplay of heterogeneous edge devices and the migration of services across devices cause a lot of security and privacy issues. Specifically, for many cloud services, their related databases/libraries usually contain edge users' sensitive information (e.g., location, account information, and medical records in real-life situations). Edge users may not trust the remote cloud server, thus unwilling to offload the computation with their private data to the server. Therefore, edge users are prone to offload their private data to their trusted nearby edge servers with privacy-preserving techniques, such as points-of-interest services and traffic information services [8].

In this context, an integrated framework is required to jointly optimize the application partitioning, resource allocation, and service caching placement for UDEC with privacy protection. However, the delay sensitivity varies for the above issues, and the complexity of the optimization problem is very high for the joint optimization problem. Recently, deep learning [9] and blockchain [10] have attracted much attention of researchers in edge computing fields. Deep learning is widely exploited in wireless communication networks (e.g., 4G and VANET) to obtain timely decision making (e.g., automatic resource allocation and robotics). Blockchain provides reliable connections and management of the computation, communication, and storage resources within the massive distributed edge nodes of UDEC [10]. Thus, there is a pressing need to explore a novel deep learning and blockchain-based scheduling approach for UDEC. In addition, federated learning (FL) [11]–[13] is regarded as a useful tool for training deep learning agents in a distributed manner, as FL can protect the personal data privacy in the model training process.

In this article, we propose an intelligent UDEC (I-UDEC) framework that jointly optimizes the issues of application partitioning, resource allocation, and service caching placement in UDEC environments.

The major contributions of this article are summarized as follows.

1) First, we propose an I-UDEC framework and formulate the heterogeneous resources (including computation, communication, and service) of the I-UDEC. We also introduce a hybrid computation offloading strategy for the I-UDEC, in which an edge user can offload its resource-intensive tasks to: a) a remote cloud server (i.e., cloud computing); b) a nearby edge server (i.e., edge computing); or c) a nearby mobile device through device-to-device (D2D) computation offloading.
2) Then, since application partitioning, resource allocation, and service caching placement have different delay sensitivities, we present a two-timescale deep reinforcement learning (2Ts-DRL) approach to jointly optimize the above issues for the I-UDEC. Specifically, the 2Ts-DRL consists of two tiers and thus operates in two different timescales. The bottom tier of the 2Ts-DRL outputs delay-sensitive decisions (i.e., application partitioning and resource allocation strategy) in a fast timescale, whereas the top tier outputs delay insensitive decision (i.e., service caching placement strategy) in a slow timescale.
3) Last but not least, we use an FL-based distributed model training method to train the deep reinforcement learning (DRL) agent, in order to protect edge users' sensitive service request information.

The remainder of this article is organized as follows. Section II presents an overview of the related works. In Section III, the architecture, system model, and computation offloading process are presented for the proposed I-UDEC framework. Section IV formulates the optimization problems and introduces our proposed 2Ts-DRL algorithm. An FL-based distributed model training method is presented in Section V. Simulation results are discussed in Section VI. Finally, conclusions are presented in Section VIII.

II. Related Work

A. Ultradense Edge Computing

To integrate cloud computing functionalities into wireless networks, European Telecommunications Standards Institute (ETSI) develops the concept of MEC [1] that is expected to play a key role in meeting the requirements of 5G. Edge servers, which are located at the edge of MEC networks, can provide computing resource, storage capability, connectivity, and various services for edge devices. However, due to the limited computation resource and storage capability of the edge servers, only a few edge devices can be served by every single server. Thus, dense deployment of edge servers in MEC is required, especially in some heavily populated urban areas (e.g., shopping malls and train stations). UDEC [4] is designed to tackle the above challenges, by deploying MEC servers on the base stations in UDN. Due to the dense deployment of the edge servers, UDEC can provide a huge computing functionalities and a variety of services close to edge users, in order to meet the low-latency demand of 5G. For example, Sun *et al.* [6] studied the mobility management issue of UDEC. They propose a new user-centric energy-aware mobility management (EMM) scheme to optimize communication and computation delay. At the same time, they also consider the long-term energy consumption constraints.

Recently, a lot of works pay much attention to the computation offloading issue of UDEC. For example, in order to reduce energy consumption, Guo *et al.* [4] proposed a greedy-based offloading scheme in a multiuser ultradense MEC server scenario. However, only one mobile device is considered in their model. Guo *et al.* [5] studied the computation offloading issue in UDEC. They propose a game-theoretical offloading scheme that can jointly be optimized task offloading, computation frequency scaling, and transmit power allocation. Thus, the energy consumption on the mobile terminal side can be minimized. However, service caching placement is ignored in this article.

B. Artificial Intelligence in Edge Computing

Recently, edge intelligence (EI) [14]–[19], aiming to integrate artificial intelligence (AI) (especially deep learning) services into edge computing, has attracted much attention. On the one hand, EI aims to push AI computations from remote cloud to the network edge, thus to achieve various real-time and reliable intelligent services. For example, Liu *et al.* [20] designed a food recognition system for dietary assessment by leveraging EI. The system operates in the edge computing environments, aims to obtain the best-in-class recognition accuracy, as well as real-time recognition. On the other hand, EI also has the potential to solve various network optimization problems (i.e., resource allocation and offloading decision). For example, He *et al.* [21] presented an integrated framework that jointly optimizes resource allocation, content caching, and computation offloading for the vehicular network. In order to solve the problem of the high complexity of the joint optimization problem, they propose a DRL-based approach to achieve automatic decision making. However, the corresponding training time is very long, the delay sensitivity for different optimization objectives (e.g., wireless resource allocation and content caching placement) is neglected.

C. Blockchain in Edge Computing

Thanks to the advantages of security, privacy, and scalability of blockchain. Integrating blockchain to edge computing has been widely studied in recent years. First, blockchain plays a key role in the secure data transmission for edge computing networks. For example, Sharma *et al.* [22] designed a blockchain-based distributed cloud architecture to provide low cost and secure access to the fog computing networks. The proposed framework also uses software-defined networking (SDN)-based controller to minimize the end-to-end delay between edge devices and system resources. Then, the decentralized P2P data storage manner of blockchain can increase the availability and scalability of edge computing networks. Since edge computing networks keep the data close to edge devices in separate storage locations. For example, Huang *et al.* [23] proposed a novel blockchain system for edge computing networks. The system stores metadata items onto blocks instead of actual data of end users by compressing the storage of blocks. They also provide a strategy that can achieve optimal places to store data and blocks. Last but not least, blockchain also enables computation offloading in edge computing networks. Huang *et al.* [23] proposed a blockchain-enabled computation offloading strategy in mobile-edge computing environments. The main objective of the strategy is to guarantee data integrity in the computation offloading process. They also use the nondominated sorting genetic algorithm III (NSGA-III) to achieve a balanced resource allocation strategy.

TABLE I
KEY NOTATION

Symbol	Definition
ED	edge device, including the 5G mobile users and IoT devices.
SCceNB	edge server, refers to the small cell cloud-enhanced e-Node B.
$\mathcal{M} = \{1, 2, ..., M\}$	The set of EDs.
$\mathcal{N} = \{1, 2, ..., N\}$	The set of SCceNBs.
C_s	The storage capacity for the SCceNBs.
f_s, f_u	The CPU frequency for each single CPU core of the SCceNBs and EDs, respectively.
$\boldsymbol{Q}_{SC}$, $\boldsymbol{Q}_{ED}$	The service caching placement index for the SCceNBs and EDs, respectively.
$\mathcal{O} = (\mathcal{V}, \mathcal{D})$	Task model, where $\mathcal{V}$ represents the set of sub-tasks, $\mathcal{D}$ denotes the data dependencies between the sub-tasks.
ξ_v	The workload (CPU cycles/number of instructions to be executed) of subtask v.
ρ_v	Required cpu cycles a CPU core will perform per byte for the input data processed by the sub-task v.
$\boldsymbol{G}_n(t)$	The task queue phase of SCceNB n in decision period t.
$\boldsymbol{E}_n(t)$	Offloading action for the task queue $\boldsymbol{G}_n(t)$ in decision period t.
$\boldsymbol{S}_n(t)$	The composite system state of SCceNB n at each decision period t.
$\boldsymbol{K}_n(t)$	Wireless resource (i.e., subcarrier) allocation strategy in decision period t.

In this article, we present a novel 2Ts-DRL algorithm. The algorithm consists of a fast-timescale and a slow-timescale learning process, in order to achieve real-time and low overhead computation offloading decisions and resource allocation strategies in a fast timescale and optimized service caching placement scheme in a slow timescale. We also leverage FL to train the 2Ts-DRL model in a distributed manner, aiming to protect the edge devices' data privacy and reduce the training overhead. In this way, the total offloading delay and network resource usage can be optimized.

III. Proposed Intelligent Ultradense Edge Computing

In this section, we will present an AI and blockchain-enhanced UDEC framework, i.e., the I-UDEC, as shown in

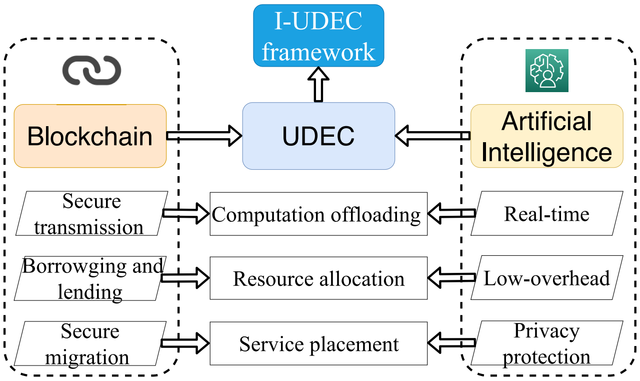

Fig. 1. AI and blockchain-enhanced UDEC framework.

Fig. 1. Specifically, we first present the architecture of the I-UDEC framework. Then, the network model and service caching model for the I-UDEC will be presented. Finally, we elaborate the computation offloading process for I-UDEC. For ease of reference, the key notations for the system model are shown in Table I.

A. Architecture of I-UDEC

The I-UDEC is realized by enhancing the following three main functions of UDEC, i.e., 1) computation offloading; 2) resource allocation; and 3) service caching placement. Computation offloading plays a critical role of UDEC, since edge devices can offload their computation-intensive tasks to their nearby resourceful edge servers. Thus, real-time offloading decision making can speed up the task execution time as well as saving the battery lifetime of edge devices. To this end, the computation offloading process is enhanced by an AI approach [i.e., deep Q-learning (DQL), as will be explained in Section IV] to achieve real-time decision making. On the other hand, during computation offloading, some security issues (e.g., jamming attacks and sniffer attacks) could be launched to disable the UDEC. Thus, secure blockchain communication (including encryption, addressing, and so on) is applied to ensure the efficient and reliable cooperation of UDEC networks. In UDEC, computation, communication, and storage resources are distributed at the network edge, thus are hard to be managed. To this end, an AI-based automatic resource allocation scheme is designed (as will be explained in Section IV) for the I-UDEC. Besides, a blockchain-based dynamic coordination mechanism (involves resource borrowing and lending) should be considered for the edge servers. Finally, an FL-based model training method is presented to protect the edge devices' data privacy. Blockchain-based service migration is considered to guarantee the security during the migration of services across multiple edge servers.

Specifically, the proposed I-UDEC framework is illustrated in Fig. 2. It consists of: 1) user plane; 2) data plane; and 3) control plane. The user plane is composed of the edge devices (EDs, such as 5G smart phones and IoT sensors) who have computation offloading requirements, as shown in Fig. 2. The data plane corresponds to: 1) remote cloud server and 2) edge servers deployed close to EDs. In addition, blockchain-based data plane cache is also applied to 1) solve the problem of saturation attacks by caching the missing packets or 2) deal with the flooding attacks by caching the flood packages. The control plane is realized by the UDN controller deployed at the macro base station (MBS). The control functionalities (i.e., monitoring, computation offloading actions, resource allocation strategies, and service caching placement) are centralized at the controller (i.e., the MBS in Fig. 2). Note that blockchain-based network monitoring [10] is applied in this plane to prevent malicious behaviors (e.g., DDoS attack and packet saturating). From a global view of the system states, the controller is responsible for collecting and maintaining the lists of EDs' information, edge server information, service information, and application information, so as to optimize the network configurations on demand. The ED information list includes data such as the computation and communication capacities for each ED. The edge server information list maintains the data of computation, storage and communication capacities, as well as the fronthaul delay for each edge server. The service information list consists of the current service caching placement strategy for the edge servers and EDs. Application information includes computation offloading requirements, which consists of the application type, required service type, data amount, and required CPU cycles. Typically, an ED reports the edge device information and application information to the nearest serving edge server, when it has computation offloading requirements. Then, the edge server integrates multiple EDs' information and edge server information together and transmits to the controller. The controller receives and trains the data through deep learning and FL modules (as will be explained in Section V). Finally, the controller makes 1) joint computation offloading and resource allocation decisions for the edge servers and EDs in a fast timescale and 2) service caching placement strategies for the edge servers in a slow timescale.

B. Network Model

In this article, we consider the small cell-based MEC [24], [25]. The edge servers of the small cell-based MEC are called small cell cloud-enhanced e-Node B (SCceNB). The EDs communicate to the SCceNBs through a wireless link, while the SCceNBs are connected to the MBS and remote cloud server via high-speed fronthaul links. Compared with the edge devices, SCceNBs are equipped with more computation and storage resources. We consider our I-UDEC framework operates in a UDEC network that formed by a remote cloud server, an MBS, a set $\mathcal{N} = \{1, 2, \dots, N\}$ of SCceNBs, and a set $\mathcal{M} = \{1, 2, \dots, M\}$ of EDs within the coverage of SCceNB n ($n \in \mathcal{N}$), as shown in Fig. 2. Assume that the ED m ($m \in \mathcal{M}$) is equipped with X_m CPU cores, each SCceNB is equipped with Y CPU cores. Thus, each SCceNB can serve at most Y EDs at the same time. Let C_s denote the storage capacities for SCceNB n. f_s and f_u represent the CPU frequency for each single CPU core of the SCceNBs and EDs, respectively.

In our I-UDEC framework, each ED can offload its tasks through establishing a cellular link with an SCceNB, or a D2D link with another ED. Assume that the cellular and D2D links

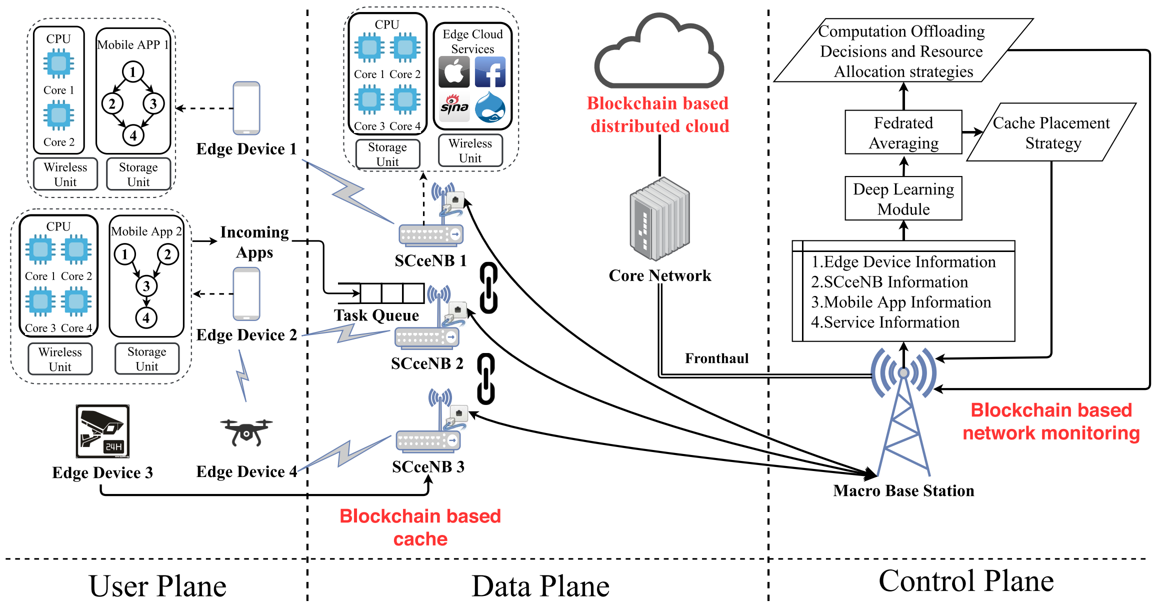

Fig. 2. Illustration of the I-UDEC framework for UDNs.

are based on the orthogonal frequency-division multiple access (OFDMA) [26]. It means that the M EDs in the coverage of SCceNB n are separated in the frequency domain, and thus do not interfere with each other. For the sake of simplicity, we utilize a subcarrier-based fixed channel assignment (FCA) scheme [27] within the M EDs. There are K available subcarriers for an SCceNB to serve EDs. The bandwidth of each subcarrier is B.

1) Cellular Link Model: During one scheduling period, assume that the locations of EDs are fixed and the wireless channels between EDs and SCceNSs are stable. Note that they may change in different scheduling periods due to EDs' mobility. Based on the above assumptions, the maximum uplink data rate (in bps) between ED m ($m \in \mathcal{M}$) and SCceNB n ($n \in \mathcal{N}$), over an additive white Gaussian noise (AWGN) channel, can be expressed as follows:

$$r^{ul}_{m,n} = K^n_m \cdot B \cdot \log_2\left(1+\frac{p^u_m|h^{ul}_{m,n}|^2}{\Gamma(g_{ul})d^{\beta}_{m,n}N_0}\right) \qquad (1)$$

where p^u_m refers to the transmit power for the ED m, $d_{m,n}$ denotes the distance between ED m and SCceNB n, and N_0 denotes the noise power. K^n_m represents that the SCceNB n allocates K^n_m subcarriers to ED m (i.e., a communication resource allocation scheme). In the Rayleigh-fading environment, $h^{ul}_{m,n}$ denotes the channel fading coefficient between ED m and SCceNB n, g_{ul} refers to the target uplink bit error rate (BER), and β is the path-loss exponent. $\Gamma(\text{BER}) = -([2\log(5\text{BER})]/3)$ is the SNR margin introduced to meet the desired target BER with a QAM constellation. In this work, we assume that these parameters are uncontrollable, similar to the assumptions that are made in [28]. Then, the uplink rate index is fed to an uplink rate weight matrix $\boldsymbol{R}^{UL}_n = [r^{ul}_{m,n}]_{1\times M}$. Note that in the UDEC network, the EDs reserve a part of resource blocks for transmitting pilot signals. The SCceNB can estimate the uplink channels by comparing the received signals and the pilot signals. Finally, the SCceNBs send the estimated channels to the controller through via high-speed fronthaul links. In practice, the channel gain can be well estimated (with small random noise). If the channel state is not perfectly estimated, our proposed DRL can actually achieve a very good approximation of the true state due to the powerful learning capability of deep learning over rich experienced learning samples, which can be verified by numerical successful application examples of DRL including intelligent wireless communications [29], [30].

2) D2D Link Model: In the proposed I-UDEC framework, each ED can offload his computation to another ED in proximity through a D2D link. For example, two edge devices ED 4 and ED 2 (as shown in Fig. 2) can offload computation to each other if their distance is less than a predefined threshold R^d, as in [31]. Let $r^{\text{d2d}}_{i,j}$ denote the D2D data rate from ED i to j as follows:

$$r^{\text{d2d}}_{i,j} = K^n_m \cdot B\log_2\left(1+\frac{p^{\text{d2d}}_i|h^{\text{d2d}}_{i,j}|^2}{\Gamma(g_{\text{d2d}})d^{\beta}_{i,j}N_0}\right) \qquad (i,j\in\mathcal{M}) \quad (2)$$

where $h^{\text{d2d}}_{i,j}$ is the channel fading coefficient for D2D link, p^{d2d}_i refers to the D2D transmission power of ED i, $d_{i,j}$ represents the distance between ED i and j, and g_{d2d} denotes the target BER for D2D link. Then, the D2D rate index is fed to a D2D rate weight matrix $\boldsymbol{R}^{\text{D2D}}_n = [r^{\text{d2d}}_{i,j}]_{M\times M}$.

In this article, we focus on the uplink energy consumption of EDs and ignore the downlink and computation energy consumption on the SCceNBs side. On the one hand, for most computation-intensive applications, the data size of outputs sent back in the downlink is much smaller than the data size of inputs in the uplink (the ratio could be as low as 1/30 [32]). On the other hand, the energy consumption of uplink transmission is about 5.5 times as much as that consumed in the downlink reception. Thus, the energy consumption of downlink reception is much less than that of uplink transmission.

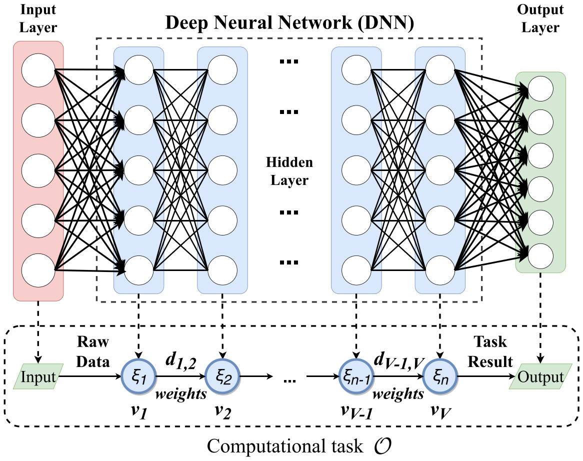

Fig. 3. Illustrative task model of the DNN model partition [33].

In addition, the SCceNBs are usually deployed in fixed areas with sufficient energy supply.

3) Service Caching Model: As stated earlier, SCceNBs and EDs have storage space to store data associated with specific computing services. Service is an abstraction of mobile applications that is managed by servers (i.e., remote cloud, SCceNBs, and EDs in this work) and requested by EDs. Running a particular service (e.g., mobile gaming, virtual reality, and Google translator) requires caching the related data (e.g., libraries and databases) on the servers. Let $\mathcal{Q} = \{1, 2, \ldots, Q\}$ represent a library which consists of Q services. Note that the services are cached at one or multiple servers (i.e., remote cloud, SCceNBs, and EDs). Each service q ($q \in \mathcal{Q}$) requires a storage space c_q. Note that the SCceNBs and EDs have limited storage capacities storing part of the popular services, whereas the remote cloud server has an infinite-capacity storage storing all the Q services. Let weight matrices $\boldsymbol{Q}_{\text{SC}} = [k_q^s]_{1\times Q}$ ($q = 1, 2, \ldots, Q$) represent the service caching placement index for the SCceNBs (i.e., all the SCceNBs share the same service caching placement scheme $\boldsymbol{Q}_{\text{SC}}$). Each element $k_q^s \in \{0, 1\}$ denotes the service caching placement index for the service q, $k_q^s = 1$ means the service q is cached in the SCceNBs, and $k_q^s = 0$ means the service k is not cached. Similarly, let weight matrices $\boldsymbol{Q}_{\text{ED}} = [k_{m,q}^u]_{M\times Q}$ ($m = 1, \ldots, M$; $q = 1, 2, \ldots, Q$) represent the service caching placement index for the EDs. Each element $k_{m,q}^u \in \{0, 1\}$ denotes the service caching placement index for the ED m, $k_{m,q}^u = 1$ means the service q is cached in the ED m, and $k_{m,q}^u = 0$ means the service k is not cached. In this work, we consider a more practical service caching placement scenario. Thus, the matrix $\boldsymbol{Q}_{\text{SC}}$ is centralized and dynamically maintained by the I-UDEC controller (i.e., the MBS), and the matrix $\boldsymbol{Q}_{\text{ED}}$ is distributed and statically managed by the EDs.

4) Task Model: In this article, we consider a fine-grained task partitioning manner [34], and split a computational task $\mathcal{O}$ into multiple subtasks. Let $\mathcal{O} = (\mathcal{V}, \mathcal{D})$, where $\mathcal{V}$ refers to the set of subtasks, and $\mathcal{D}$ denotes the data dependencies between the subtasks. Assume that V represents the number of subtasks in task $\mathcal{O}$ (i.e., $V = |\mathcal{V}|$). Each subtask v ($v \in \mathcal{V}$) can be: 1) offloaded to a cloud server; 2) offloaded to an edge server; or 3) executed locally in the edge device. Note that the outputs of some subtasks are the inputs of others, thus, the dependencies has a great influence on the task execution and computation offloading, and cannot be ignored. Generally, there exist three typical dependency models: 1) sequential; 2) parallel; and 3) general dependency [35]. Take the task of the deep neural network (DNN) model partition [33] with sequential dependency as an example. In order to reduce the burden of EI applications execution on end devices, the DNN model partition can split a DNN training phase (i.e., the computational task $\mathcal{O}$) into multiple subtasks (i.e., the hidden layers of DNN), as shown in Fig. 3. By dynamically offloading the resource-intensive subtasks to the edge server, the DNN model partition can speed up the inference process. In this work, we consider the sequential dependency task model in our framework.

Let a parameter tuple $k_v = \langle v, \xi_v, d_{u,v}, d_{v,w}, q(v)\rangle$ characterize the subtask v of task $\mathcal{O}$, where v is the current subtask, u and w refer to the previous subtask and next subtask of v, ξ_v ($v \in \mathcal{V}$) denotes the workload of subtask v, and $q(v)$ represent the required service of subtask v. $d_{u,v}$ and $d_{v,w}$ refer to the input and output data size of subtask v (e.g., weights of the DNN model), respectively. Let ρ_v (in CPU cycles/byte) denote the required CPU cycles, a CPU core will perform per byte for the input data processed by the subtask v (i.e., the complexity of subtask v). Thus, we have $\xi_v = \rho_v \cdot d_{u,v}$.

C. Computation Offloading in I-UDEC

In this article, the process of computation offloading in the I-UDEC framework consists of the following phases: 1) service caching placement; 2) computation offloading decisions; and 3) resource allocation. First, due to the limited storage capabilities of SCcsNBs, an optimal service caching placement strategy for the SCcsNBs is required to shorten the service distance between EDs and their serving SCcsNBs. However, unlike the remote cloud which is equipped with huge computation and storage resources, an individual SCceNB with limited resource allows only part of the services to be cached at a time. Thus, the key problems for service caching in I-UDEC are 1) how to estimate the popularity of services and cache the popular services and 2) how to balance the tradeoff between multiple services and finite storage capacity of SCceNBs. Second, a fine-grained application partitioning strategy is used to reduce task execution time and improve quality-of-service (QoE) for EDs. Fine-grained application partitioning divides a mobile application into multiple subtasks, and dynamically offloads only the computation-intensive subtasks of the application. Note that real time and dynamic partitioning allows mobile devices to obtain the highest benefit through computation offloading, thus plays a critical role in UDEC networks. Finally, SCceNBs need to allocate computation, communication, and storage resources to the offloaded subtasks. After partitioning mobile applications, SCceNB needs to allocate available system resources to process the offloaded parts. Thus, an efficient resource allocation scheme will minimize the system resource usage, and significantly improve the experience of edge devices. To this end, it makes sense to jointly optimize the service caching placement, application partitioning, and resource allocation for the I-UDEC.

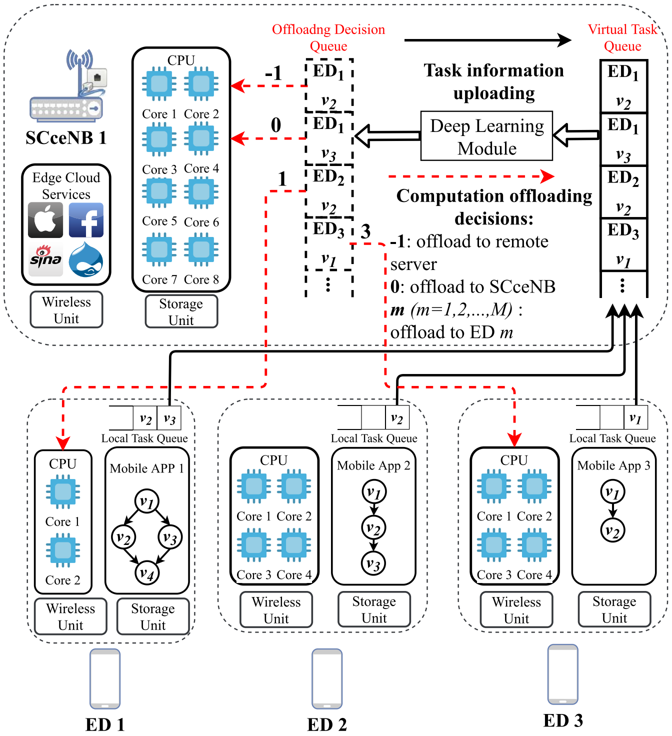

Fig. 4. Computation offloading process for I-UDEC.

To this end, the computation offloading process for the I-UDEC framework is illustrated in Fig. 4. In this work, we consider a hybrid computation offloading scenario for the I-UDEC, which means that an ED can either execute subtasks locally or offload computation-intensive subtasks to 1) another ED through the D2D link; 2) an SCceNB through wireless uplink; or 3) remote cloud server through the wireless uplink and high-speed fronthaul. Furthermore, we assume that the offloading decision making operates in a time-slot structure, and the decision making period is discrete time frames $t \in \mathcal{T} = \{0, 1, 2, \ldots, T\}$.

In the I-UDEC framework, we consider each ED is served by its closest SCceNB. Each SCceNB manages a virtual task queue to store the information of the subtasks pending for execution and outputs a queue of offloading decisions for edge devices, as shown in Fig. 4. Let matrix $\boldsymbol{G}_n(t) = \{\boldsymbol{g}_1(t), \boldsymbol{g}_2(t), \ldots, \boldsymbol{g}_l(t)\}$ represent the task queue phase of SCceNB n in decision period t, where l denotes the maximum queue length for the task queue. $\boldsymbol{g}_i(t) = \{m, k_v\}$ denotes a parameter vector for the ith element in the task queue.

Define $\boldsymbol{E}_n(t) = \{e_1(t), e_2(t), \ldots, e_l(t)\}$ as the computation offloading action for the task queue $\boldsymbol{G}_n(t)$ in decision period t, where $e_i(t)$ ($e_i(t) \in \{-1, 0, 1, 2, \ldots, l\}$) represents the offloading action for $\boldsymbol{g}_i(t)$, as shown in Fig. 4. $e_i(t) = -1$ denotes to offload the subtask v in $\boldsymbol{g}_i(t)$ to the remote cloud server, $e_i(t) = 0$ represents to offload the subtask to the serving SCceNB, and $e_i(t) = j$ ($j = 1, 2, \ldots, M$) indicates to offload the subtask to the ED j. Note that $m = j$ represents the local execution.

Based on the above assumptions, we can obtain the total execution time of the subtasks in the current task queue in $\boldsymbol{G}_n(t)$ as follows:

$$T^{\text{exe}}(t) = \sum_{i=1}^{l} t_i^{\text{exe}}(t) \tag{3}$$

where t_i^{exe} represents the execution time for the ith element $\boldsymbol{g}_i(t) = \{m, k_v\}$ in the current task queue $\boldsymbol{G}_n(t)$ as follows:

$$t_i^{\text{exe}}(t) = \begin{cases} \frac{\rho_v \cdot d_{u,v}}{X_m \cdot f_u}, & e_i(t) = j;\ 1 \le j \le M; j = m \\ \frac{\rho_v \cdot d_{u,v}}{X_j \cdot f_u} + \frac{d_{u,v}}{r_{m,j}^{\text{d2d}}}, & e_i(t) = j; 1 \le j \le M; j \ne m \\ k_{j,q(v)}^{u} = 1 & \\ \frac{\rho_v \cdot d_{u,v}}{f_s} + \frac{d_{u,v}}{r_{m,n}^{ul}}, & e_i(t) = 0; k_{q(v)}^{s} = 1 \\ \frac{d_{u,v}}{r_{m,n}^{ul}} + t_n^e, & e_i(t) = -1,\ \text{else}. \end{cases} \tag{4}$$

Note that $k_{j,q(v)}^{u} = 1$ guarantees the requires service $q(v)$ of subtask v is cached in ED j, if ED m decides to offload his subtask subtask v to ED j through D2D offloading. Similarly, $k_{q(v)}^{s} = 1$ guarantees requires service $q(v)$ of subtask v is cached in SCceNB n, if ED m performs edge offloading (i.e., offload his subtask to the nearby SCceNB n). t_n^e refers to the end-to-end latency between SCceNB n and the remote cloud server, when ED m performs cloud offloading.

IV. Problem Formulation and Algorithm Design

In this section, we will formulate the optimization problem of joint application partitioning, resource allocation, and service caching placement in the I-UDEC framework. The main objective of the optimization problem is to jointly minimize the tasks' long-term execution time and system resources' usage (i.e., computation, communication, and storage usage). To this end, we utilize the DQL algorithm to tackle the formulated optimization problem and present a novel 2Ts-DRL approach to train the decision agent, as shown in Fig. 5.

In reinforcement learning (RL) [9], an agent can periodically learn to take actions, observes the most reward, and automatically adjusts its strategy in order to obtain the optimal policy. For example, Q-learning [14] is a widely used model-free RL algorithm in the literature. In Q-learning, decision agent can reach to the best policy by estimating the Q-function (be defined as the state-action function). Q-function utilizes the samples which are obtained during the interactions with the environment to approximate the values of state–action pairs. However, it is time consuming for RL to obtain the best policy, since RL must explore and gain knowledge of the whole environment. Thus, RL is unsuitable and inapplicable to large-scale networks, especially for the UDN environment in the I-UDEC. Deep learning is introduced as a state-of-the-art technique to tackle the above limitation of RL, and opens a novel research field, namely, DRL. DRL improves the learning speed and RL performance by leveraging DNNs to train the learning model.

The DQL algorithm which integrates deep learning into RL, is introduced to solve the problem of small state space and action space in Q-learning. The DQL implements a deep

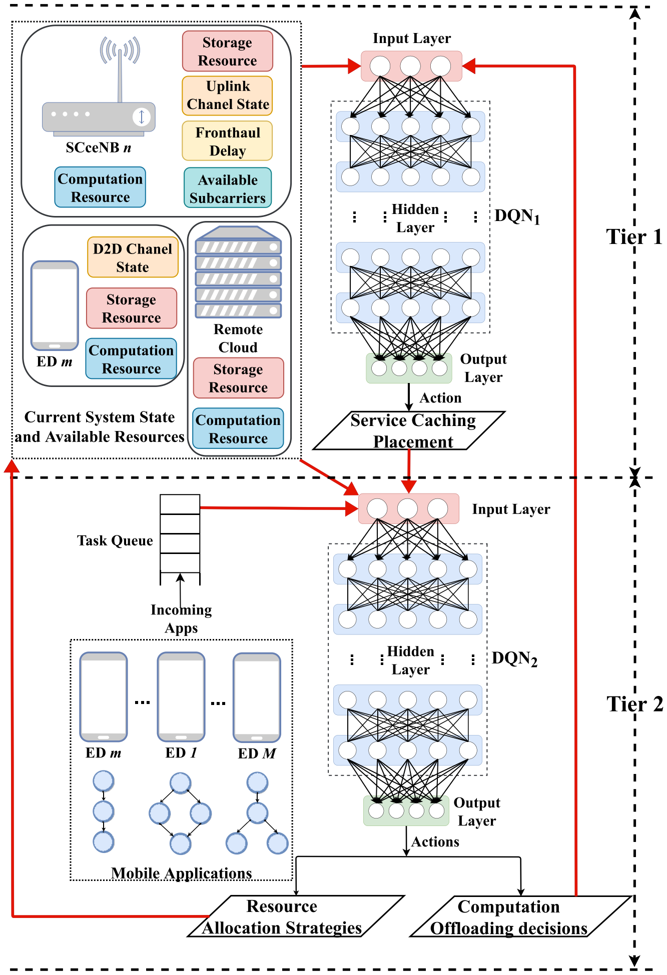

Fig. 5. Proposed 2Ts-DRL framework.

Q-network (DQN) instead of the Q-table of Q-learning, thus, the learning speed is significantly improved.

Note that, in the I-UDEC framework, the application partitioning, resource, and service caching placement strategies have different delay sensitivities. To this end, we present a novel 2Ts-DRL method to jointly optimize the above issues in two different timescales. Specifically, we design a two-tier structure DQN agent for the DRL, which operates in two different timescales, as shown in Fig. 5. The bottom tier of the agent outputs delay-sensitive decisions (i.e., application partitioning and resource allocation strategies) in a fast timescale, whereas the top tier outputs delay insensitive decision (i.e., service caching placement strategy) in a slow timescale.

Specifically, we model the problem as a Markov decision process (MDP) [36] and formulate the state space, action space, and reward function of the MDP as follows.

A. State Space

At the beginning of each decision period t, the EDs send the information of their computation offloading requests, current service caching placement strategy, and available resource to their nearby SCceNBs and the controller (i.e., the MBS in Fig. 2). Then, each SCceNB receives the uploaded information and builds a virtual task queue $\boldsymbol{G}_n(t)$ $(n \in \mathcal{N})$. Thus, the system state of SCceNB n at each decision period t is composed of the task queue state and the available resource state, which is given as follows:

$$\begin{aligned}\boldsymbol{S}_n(t) = \Big\{&K(t), \boldsymbol{R}_n^{UL}, \boldsymbol{R}_n^{\mathrm{D2D}}, \boldsymbol{Q}_{\mathrm{SC}}(t), \boldsymbol{Q}_{\mathrm{ED}}(t), \boldsymbol{G}_n(t), \boldsymbol{X}_n(t)\\ &\boldsymbol{Y}_n(t), \boldsymbol{Z}_n(t)\Big\}\end{aligned} \tag{5}$$

where $K(t)$ denotes the available subcarriers to be allocated at decision period t. $\boldsymbol{R}_n^{UL}$ and $\boldsymbol{R}_n^{\mathrm{D2D}}$ are the uplink and d2d data rate weight matrices, respectively. $\boldsymbol{Q}_{\mathrm{SC}}(t)$ and $\boldsymbol{Q}_{\mathrm{ED}}(t)$ denote the service caching placement state for the SCceNB n and M EDs at decision period t, respectively. $\boldsymbol{G}_n(t)$ refers to the current task queue state. $\boldsymbol{Y}_n(t) = \{y_1(t), y_2(t), \ldots, y_Y(t)\}$ is the current computation resource (i.e., CPUs) state, where $y_i(t) = 1$ $(i = 1, 2, \ldots, Y)$ represents the ith CPU core of SCceNB n is available, and $y_i(t) = 0$, not available. Similarly, $\boldsymbol{X}_n(t) = \{x_1(t), x_2(t), \ldots, x_M(t)\}$ is the current computation resource the M EDs within the coverage of SCceNB n, where $x_j(t) = 1$ $(j = 1, 2, \ldots, M)$ represents the computation resource of the jth ED is available, and $x_j(t) = 0$, not available. $\boldsymbol{Z}_n(t) = \{q_v, v \in \boldsymbol{G}_n(t)\}$ represent the set of required services for the subtasks in the task queue.

B. Action Space

In the I-UDEC framework, the decision agent has to take the following three actions: 1) application partitioning actions $\boldsymbol{E}_n(t)$ for the current task queue $\boldsymbol{G}_n(t)$; 2) the wireless resource (i.e., subcarrier) allocation strategy $\boldsymbol{K}_n(t) = \{K_1^n(t), K_2^n(t), \ldots, K_M^n(t)\}$, where $K_m^n(t)$ $(m = 1, 2, \ldots, M)$ represents the number of subcarriers that allocates to the ED m at decision period t; and 3) the service caching placement policy $\boldsymbol{Q}_{\mathrm{SC}}(t)$ for the SCceNBs. Note that $\boldsymbol{E}_n(t)$ and $\boldsymbol{K}_n(t)$ are delay-intensive actions, which must be made immediately (i.e., real-time resource scheduling). Whereas $\boldsymbol{Q}_{\mathrm{SC}}(t)$ is a delay insensitive action, because SCceNBs do not have to frequently update their cached services. Thus, we assume a time interval αt for SCceNB n to update their cached services. It means that the SCceNBs update their service caching strategy $\boldsymbol{Q}_{\mathrm{SC}}(t)$ every αt seconds.

C. Reward Function

In order to minimize the overall task execution time for the EDs and reduce the system resource usage of I-UDEC, we consider a comprehensive revenue of the I-UDEC framework as the system's reward. The revenue combines of the computation, communication, and storage resources usage, as well as the task execution time for the EDs.

For the delay insensitive service caching placement strategy (i.e., tier 1 in Fig. 5), given the system state $\boldsymbol{S}_n(t)$, service request history $\boldsymbol{Z}_n(t - \alpha t, t - 1)$, current service caching placement strategies $\boldsymbol{Q}_{\mathrm{SC}}(t)$, $\boldsymbol{Q}_{\mathrm{ED}}(t)$, and resource allocation strategy $\boldsymbol{K}_n(t)$ at current decision period t, the immediate reward $R_s(t)$ of service caching can be expressed as follows:

$$\begin{aligned}R_c(t) &= \zeta_1 \frac{T^{\mathrm{exe}}(t)}{T^{\mathrm{loc}}(t)} + \zeta_2 \frac{\sum_{q=1}^{Q} k_q^s}{C_s}\\ \text{s.t. } &T^{\mathrm{exe}}(t) \le T^{\mathrm{loc}}(t)\\ &\sum_{q=1}^{Q} k_q^s \le C_s\end{aligned} \tag{6}$$

where ζ_1 and ζ_2 denote weight factors. $T^{\text{loc}}(t)$ is the total local execution time for the subtask in the current task queue as follows:

$$T^{\text{loc}}(t) = \sum_{i=1}^{l} \frac{\rho_v \cdot d_{u,v}}{X_m \cdot f_u} \qquad v \in \boldsymbol{g}_i(t). \tag{7}$$

Note that we use the service request history $\boldsymbol{Z}_n(t-\alpha t, t-1)$ to estimate the popularity of the services, and the SCceNBs prone to cache the most popular services to enhance their cache hit probability. The first term of $R_c(t)$ in (6) is a normalized execution time index, and the first constraint of (6) guarantees the total execution time through computation offloading must less than the time of local processing. Similarly, the term second of $R_c(t)$ in (6) represents a normalized storage usage index, and the second constraint of (6) guarantees the total storage consumption of current service caching strategy $\boldsymbol{Q}_{\text{SC}}(t)$ must less than the storage capacities of the SCceNBs. The objective of the service caching placement is to minimize the long-term execution time for the EDs as well as the storage usage for the SCceNBs. Since the objective of DRL is to maximize reward, the value of long-term reward should be negatively correlated to the immediate reward as follows:

$$R_c^{\text{long}} = \max \sum -R_c(t), \qquad t = 0, \alpha t, 2\alpha t \quad \ldots, T. \tag{8}$$

For the application partitioning and resource allocation strategies (i.e., tier 2 in Fig. 5), given the system state $\boldsymbol{S}_n(t)$, current service caching placement strategies $\boldsymbol{Q}_{\text{SC}}(t)$, $\boldsymbol{Q}_{\text{ED}}(t)$, available communication resource $K(t)$, available computation resource $\boldsymbol{Y}_n(t)$ and $\boldsymbol{X}_n(t)$ at current decision period t, and the immediate reward $R_a(t)$ of application partitioning and resource allocation can be expressed as follows:

$$R_a(t) = \mu_1 \frac{T^{\text{exe}}(t)}{T^{\text{loc}}(t)} + \mu_2 \frac{\sum_{m=1}^{M} K_m^n(t)}{K(t)} + \mu_2 \frac{\sum_{j=1}^{l} e_j(t) = 0}{\sum_{i=1}^{Y} y_i(t)} \tag{9}$$

where μ_1, μ_2, and μ_3 denote weight factors. The first term in (9) represents a normalized execution time index, as shown in (6). The second term in (9) denotes a normalized communication resource usage of SCceNB n, which guarantees the number of consumed subcarriers must less than the number of available subcarriers at decision period t. The third term in (9) is a normalized computation resource usage of SCceNB n, which guarantees the number of occupied CPUs must less than the number of available CPUs of SCceNB n at decision period t. By leveraging DQN, we can obtain an optimal policy to maximize the long-term reward, the cumulative reward should be negatively written as follows:

$$R_a^{\text{long}} = \max \sum -R_a(t) \quad t = 0, 1, 2, \ldots, T. \tag{10}$$

Note that (8) and (10) have a similar composition, but they operate in different timescale.

Finally, our decision agent will output near-optimal resource allocation strategies, computation offloading decisions, and service caching placement strategies to minimize the long-term cost. Service caching allows for a long updating interval while computation offloading needs a fast response.

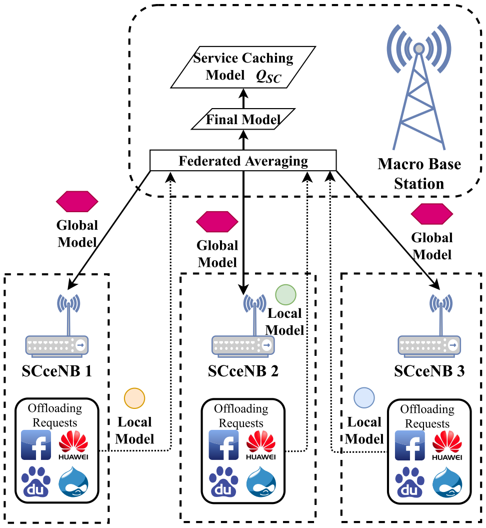

Fig. 6. FL-based model training.

V. Federated-Learning-Based Model Training

For the service caching placement agent of the proposed two-tier DQN (i.e., tier 1 in Fig. 5), it is critical to predicting the future popularity of services. Thus, the most popular ones can be cached to minimize the long-term reward in (8). Thus, the controller (i.e., the MBS in Fig. 2) collects the service request history $\boldsymbol{Z}_n(t-\alpha t, t-1)$ to make the prediction. However, it may bring a significant challenge: EDs may not trust the controller and thus unwilling to transmit the information of their offloading requests (i.e., most used services).

To this end, we design an FL-based model training method in order to distributively train the service caching placement agent, as shown in Fig. 6. Algorithm 1 shows the pseudocode of the proposed algorithm. Specifically, at the beginning of the decision period t, all the N SCceNBs first receive the global model $\boldsymbol{W}(t)$ (i.e., DRL weights) according to their current service placement strategy $\boldsymbol{Q}_{\text{SC}}(t)$ from the central controller. Then, the SCceNBs compute the local models $\boldsymbol{W}_1(t), \boldsymbol{W}_2(t), \ldots, \boldsymbol{W}_N(t)$ based on their current service request $\boldsymbol{Z}_n(t)$, $(n = 1, 2, \ldots, N)$. Next, each SCceNB n uploads their updated model [be written as $\boldsymbol{H}_n(t) := \boldsymbol{W}(t) - \boldsymbol{W}_n(t)$] to the central controller. The controller receives and aggregates the updates, and then constructs an updated global model $\boldsymbol{W}(t+1)$ in the next decision period by using federated averaging [9], [37] as follows:

$$\boldsymbol{W}(t+1) = \boldsymbol{W}(t) + \psi \boldsymbol{H}(t), \boldsymbol{H}(t) := \frac{1}{N} \sum_{n=1}^{N} \boldsymbol{H}_n(t) \tag{11}$$

where ψ represents the learning rate.

The centralized controller aggregates all updated models by utilizing the weighted average sum. At the same time, the controller considers the quantity of each SCceNB's local data set. The iteration repeats until it reaches to the next service caching placement decision period $t + \alpha t - 1$. The controller generates a service caching placement strategy $\boldsymbol{Q}_{\text{SC}}(t + \alpha t)$ for the next caching placement decision period $t + \alpha t$ and broadcasts

Algorithm 1 FL-Based Model Training for Service Caching Placement

```
Initialization:
Controller side:
    Initialize the DRL model with random weights $\boldsymbol{W}(0)$ at the beginning of decision period $t = 0$.
SCceNBs' side:
    Initialize the local DRL model weights $\boldsymbol{W}_n(0)$, $(n = 1, 2, \ldots, N)$;
    Download $\boldsymbol{W}(0)$ from the controller and let $\boldsymbol{W}_n(0) = \boldsymbol{W}(0)$, $(n = 1, 2, \ldots, N)$.
Iteration: for each decision period $t = 0$ to $T$ do
function FL($\boldsymbol{Z}_n(t)$)
    SCceNBs' side:
    while $t > 0$ do
        for each SCceNB $n \in \mathcal{N}$ in parallel do
            download $\boldsymbol{W}(t)$ from the controller;
            let $\boldsymbol{W}_n(t) = \boldsymbol{W}(t)$;
            train the DRL agent locally with $\boldsymbol{W}_n(t)$ on the current service requests $\boldsymbol{Z}_n(t)$;
            Upload the trained weights $\boldsymbol{W}_n(t+1)$ to the controller;
        end for
    Controller side:
        receive all weights $\boldsymbol{W}_n(t)$ updates;
        perform federated averaging;
        broadcast averaged weights $\boldsymbol{W}_n(t + 1)$;
    end while
end function
```

the strategy to the SCceNBs. Then, the SCceNBs update their local cached services according to the received caching placement strategy and fetch the services from the backbone network. Finally, the SCceNBs enter the next decision period and train the fast timescale agents (as shown in Fig. 5) asynchronously in parallel upon the current system state.

In addition, we use asynchronous advantage actor–critic (A3C) [38] to train the agents, owing to its asynchronous advantage. Under the field of DRL, A3C uses asynchronous multithreads to train DNN through multiple agents. The agents are controlled by a global network, can learn and run asynchronously, thus keep their own network parameters and the copies of the environment with different policies. Each time the agents first update their own parameters of the actor network and the critic network (i.e., the local model in Fig. 6), and also a global actor network and a global critic network (i.e., the global model in Fig. 6). Then, they submit the parameters to the global networks. Finally, the global network receives and distributes the received parameters. The local and global networks will converge after multiple iterations. This kind of training has the following advantages: 1) lower memory usage: since the replay memory (e.g., Q-table) of standard DRL is not needed; 2) avoiding the correlation: since different agents will likely experience different states and transitions; and 3) faster and more robust than the traditional DRL approach.

VI. Performance Evaluation

In this section, the performance of the proposed 2Ts-DRL is evaluated through computer simulations. Specifically, we use the MATLAB RL toolbox and the deep learning toolbox [39] to implement the 2Ts-DRL framework. First, we describe the simulation environment and introduce the related benchmark strategies for the proposed I-UDEC. Then, we compare the performance of the proposed 2Ts-DRL algorithm with the benchmark strategies and discuss simulation results. The values of the parameters are summarized in Table II.

TABLE II
Network Parameters

Parameter	Value	Parameter	Value
B	15KHz	μ_1, μ_2, μ_3	1/3
M	5	ζ_1, ζ_2	1/2
N	15	Q	30
f_u	10^8 cycles/s	c_q	[100, 300]M
f_s	10^9 cycles/s	C_s	2G
ρ_v	[4000, 12000] cpb	$d_{u,v}$	[100, 500] KB
Learning rate	0.001	Discount factor	0.95
Optimizer	Adam	Activation function	ReLU
Exploration rate	0.1	Hidden layer	3

A. Simulation Environment

In our simulation, we consider a UDEC network environment. For the communication resource in the network, we assume that the bandwidth of each subcarrier B is 15 kHz, and each SCcenNB has 256 subcarriers to be allocated (i.e., $K = 256$). For the computation resource, we assume that the computation capacity f_u (i.e., CPU frequency) of a single CPU core for the EDs is 10^9 cycles/s, and the number of CPU core X_m for ED m is uniformly selected from the set $\{1, 2, 4, 8\}$. Similarly, let the computation capacity f_s of a single CPU core for the SCceNBs is 10^9 cycles/s, and the number of CPU core Y for the SCceNBs is fixed to 8. For the services required by the EDs, we assume that there are 30 kinds of services (i.e., $Q = 30$) and the data size of each service c_q follows the uniform distribution in the range of [100, 300] M. We also consider a popularity-based caching placement policy [40], thus the request probability for a service q ($q \in \mathcal{Q}$) is

$$f(q, \delta, Q) = \frac{1}{q^\delta} \sum_{n=1}^{Q} \frac{1}{n^\delta} \tag{12}$$

where δ represents the skewness of the popularity profile.

Afterward, we implemented our proposed 2Ts-DRL-based task execution scheme and compare it with respect to the following four task execution schemes, a resource allocation strategy and a service caching placement policy.

1) *Local Execution Scheme (LES):* Local execution, all the subtasks are locally executed on the edge devices.
2) *Edge Execution Scheme (EES):* The EDs offload all their tasks to their nearby SCceNBs for edge processing. Note that if the required service for the subtask v of ED m is not cached in a nearby SCceNBs. ED m processes the subtask v locally.
3) *Random Execution Scheme (RES):* RES represents a random computation offloading strategy, where each subtask is randomly offload to SCceNB, cloud server, a nearby ED, or be executed locally.
4) *Cloud Execution Scheme (CES):* All the subtasks are offloaded to remote cloud server.

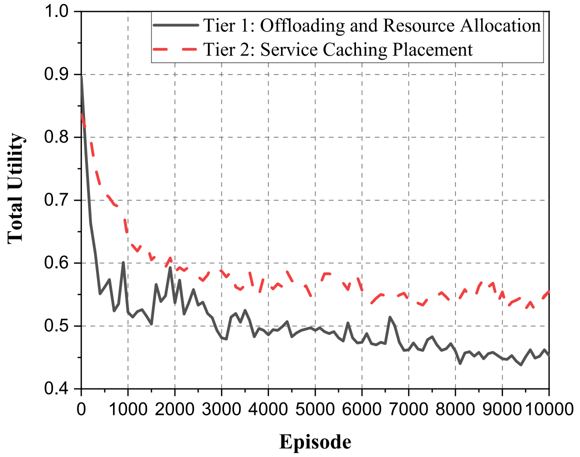

Fig. 7. Convergence performance of the proposed 2Ts-DRL.

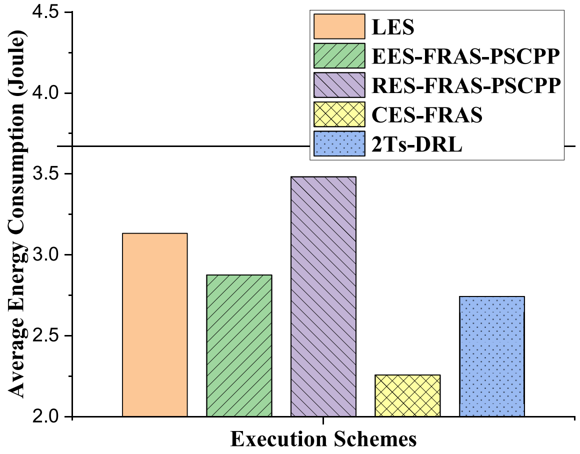

Fig. 9. Energy consumption for different offloading schemes when wireless link is intermittent.

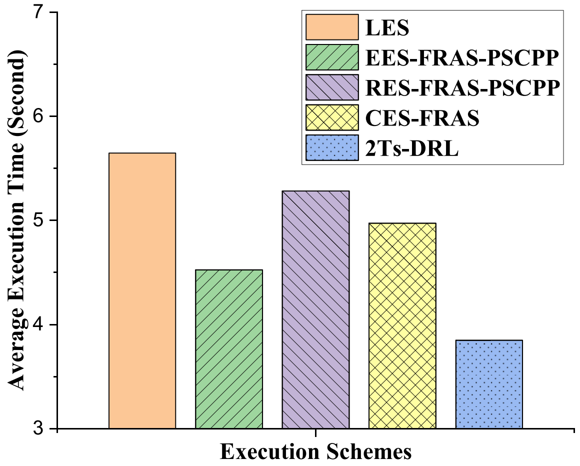

Fig. 8. Task execution time for different execution schemes.

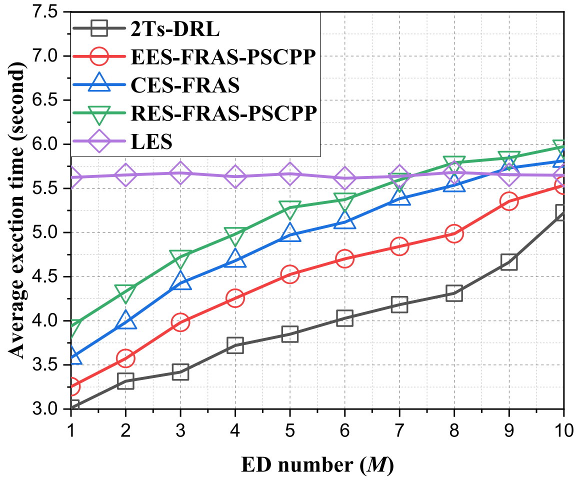

Fig. 10. Average task execution time versus ED density M.

5) *Fair Resource Allocation strategy (FRAS):* The communication resource (i.e., available subcarriers) is equally shared by the EDs.
6) *Popularity-based Service Caching Placement Policy (PSCPP):* The SCceNBs always cache the most popular services until reaching to their storage capacity C_s.

B. Simulation Results

Fig. 7 illustrates the convergence performance for the 2Ts-DRL algorithm. Several observations can be made. First, we can find that the total utility for both of the tiers of the 2Ts-DRL is high at the beginning of the DRL process. The utility decreases when the number of episodes increases. Then, tier 1 of the 2Ts-DRL converges faster than the tier 2 of the 2Ts-DRL, since tier 1 has smaller state and action spaces.

Fig. 8 reports the average long-term execution time of the subtasks under different execution schemes. Since resource allocation strategy and service caching placement policy have no impact on the LES, and service caching placement policy has no impact on the CES. We compare the performance of our proposed 2Ts-DRL with the benchmark strategies that is illustrated in Fig. 8. Specifically, EES-FRAS-PSCPP refers to an EES with FRAS and PSCPP. RES-FRAS-PSCPP denotes an RES with FRAS and PSCPP. CES-FRAS represents a CES with FRAS when EDs upload tasks to their nearby SCceNBs. Note that the 2Ts-DRL outperforms other execution schemes in execution time saving. In fact, our 2Ts-DRL can reduce the task execution time on average by 31.87%, 14.96%, 27.16%, and 22.61% compared to the LES, EES-FRAS-PSCPP, RES-FRAS-PSCPP, and CES-FRAS schemes, respectively.

Fig. 9 illustrates the average energy consumption of the subtask execution. Note that we only consider the energy consumption on the edge devices and ignore the execution energy consumption on the SCceNBs and remote cloud server. Thus, CES-FRAS consumes the least amount of energy, since all the subtasks are offloaded to the remote cloud server and EDs only consume the energy for data transmission. However, due to the long end-to-end delay, the execution time is long with respect to the 2Ts-DRL, as shown in Fig. 8. Note that our 2Ts-DRL outperforms EES-FRAS-PSCPP and RES-FRAS-PSCPP and achieves 25.15%, 17.25%, and 10.57% energy reduction compared to the above execution schemes.

Fig. 10 illustrates the average task execution time for different ED density M to evaluate the performance of our proposal in the UDEC scenario. First, our proposal 2Ts-DRL outperforms the related benchmark policies in terms of task execution time. Note that the LES has stable task execution time when the

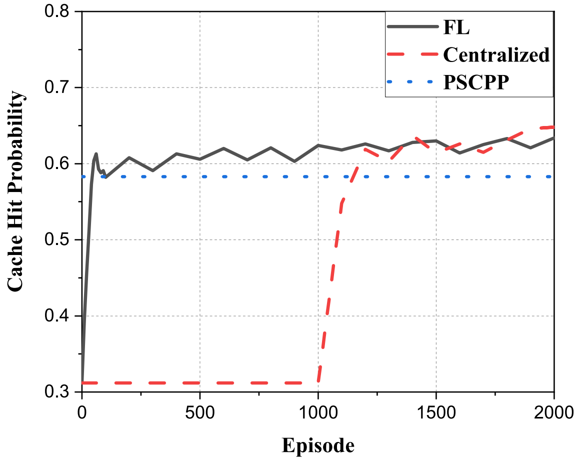

Fig. 11. Cache hit probabilities for different service caching placement policies.

density grows from 1 to 10, since LES performs local execution, has no additional transmission delay. The execution time of RES-FRAS-PSCPP, CES-FRAS, EES-FRAS-PSCPP, and our proposed 2Ts-DRL increases when the ED density grows. Because all the EDs within the same SCceNB coverage share the limited computation and communication resources. Thus, the performance of the task execution time declines as the number of ED increases. However, the problem can be solved by deploying more SCceNBs at the network edge, especially in some hotspots areas. For our 2Ts-DRL, the average task execution time increases dramatically when $M > 8$. The reason is that each SCceNB has a limited computation capacity, can serve up to eight EDs in our simulation. Thus, some EDs have to perform LES when $M > 8$.

In this work, we utilize the cache hit probability as the performance metric to evaluate our proposed 2Ts-DRL. The metric is the ratio of service cache hits to the number of ED's service requests on the cache. Fig. 11 reports the cache hit probabilities for different service caching placement policies. Specifically, FL refers to the proposed FL-based model training policy. Centralized represents a centralized model training policy, which means that the decision agent uses the service requests from the last $1000t$ decision period. Several observations can be made. First, PSCPP has a fixed cache hit probability, since all the services have predefined popularity indexes, and PSCPP always cache the most popular services. Then, the centralized has a fixed cache hit probability from the decision period 0 to 1000 when the decision agent collects the service requests from the EDs. The decision agent begins to train the model at a decision period $t = 1000$, thus the cache hit probability grows with the number of the episodes increases. Finally, we can find that the cache hit performance of our proposed FL is near close to the results of centralized. However, our proposed FL-based model training method can protect the data privacy (e.g., most used type of services) for the EDs.

Fig. 12 reports the cache hit probabilities of different service caching placement policies under different popularity index δ. All the policies have a similar cache hit probabilities when $\delta = 0$, since the popularity profile is uniform over all the services if $\delta = 0$. The cache hit probability grows with δ increases for the three policies, and DRL-based training policies (both centralized and distributed) always outperform PSCPP in the cache hit probability.

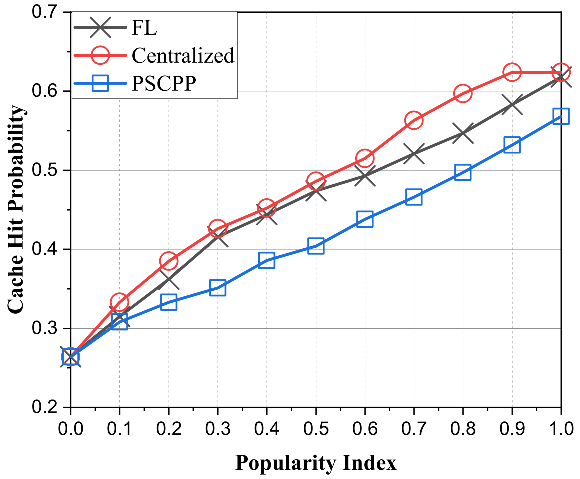

Fig. 12. Cache hit probabilities versus popularity index δ.

VII. Extension and Future Works

In this section, we will discuss some potential and interesting research directions to extend this work.

A. *Blockchain-Enabled I-UDEC*

In recent years, blockchain, as the underlying technology of cryptocurrencies, has been adopted in various applications, and attracted much attention. Note that our proposed I-UDEC still faces some challenges, such as decentralized resource management and security. Integrating blockchain into I-UDEC is synergistically beneficial for the following reasons. First, security and privacy are significant challenges to I-UDEC due to the interplay of heterogeneous edge nodes (i.e., SCceNBs and edge devices) and the attack vulnerability of the edge nodes. In this work, we propose an FL-based 2Ts-DRL algorithm for the I-UDEC to guarantee the privacy of edge devices. Note that the I-UDEC still faces the security challenges, such as DDoS attack and packet saturating. It makes sense to integrate blockchain into I-UDEC, since the integration can 1) replace the expensive key management for multiple communication protocols; 2) enable easy access for the maintenance of the distributed ultradense edge SCceNBs of I-UDEC; and 3) provide efficient monitoring in the control plane to prevent malicious behaviors. Second, in order to tackle the challenge of resource heterogeneity of the I-UDEC network, a distributed resource management scheme is required. It is possible to build a distributed control at massive SCceNBs by leveraging blockchain, since blockchain has the feature of decentralized control. The resource management involves edge resource borrowing and lending, its pricing-based optimization algorithms will play a key role in resource overhead reduction. Specifically, the smart contract of blockchain enables the use of edge resources (i.e., computation, communication, and storage resources) on demand. Because smart contract can automatically run on-demand resource algorithm for the computation offloading requirements. Finally, blockchain systems are hungry for hardware resource (e.g., computing power), and I-UDEC has rich network resources at the network edge. Thus, the deployment of the blockchain in the I-UDEC is mutually beneficial. The above research directions will be concerned in our future work.

B. Personalized Federated-Learning-Enhanced 2Ts-DRL

Note that in practical scenarios, SCceNBs in different areas may have different service caching placement states (i.e., service popularity distribution). As a result, it is hard to aggregate the local models with different shapes through traditional FL. To this end, we can use personalized FL (PFL) [12], [41] to capture the characteristics of local service popularity and local offloading requirements. In PFL, a global model is first trained by standard FL, then, each SCceNB will train a personalized model based on the global model information and its own personal information. Thus, each SCceNB has a personalized service caching model for different EDs tailored to their computation offloading requirements. This kind of heterogeneous service caching placement scenario will be studied in our future work.

VIII. Conclusion

In this article, a joint computation offloading, resource allocation, and service caching placement problem is studied for UDEC networks. First, we introduced an I-UDEC framework in 5G UDN environments, in order to formulate the heterogeneous network resources and hybrid computation offloading pattern of UDEC. Then, in order to minimize the task execution time and network resource usage, we optimize the application partitioning, resource allocation, and service caching placement in two different timescales. To this end, we present a Ts-DRL approach to jointly optimize the above issues for the I-UDEC. The bottom tier of the 2Ts-DRL outputs delay-sensitive decisions in a fast timescale, whereas the top tier outputs delay insensitive decision in a slow timescale. Last but not least, we use an FL-based distributed model training method to train the 2Ts-DRL model, in order to protect edge users' sensitive service request information. Experimental results corroborate the effectiveness of both the 2Ts-DRL and FL in the I-UDEC framework.

References

[1] ETSI. (2018). *Multi-Access Edge Computing (MEC)*. [Online]. Available: https://www.etsi.org/technologies-clusters/technologies/multi-access-edge-computing

[2] X. Wang, X. Li, S. Pack, Z. Han, and V. C. M. Leung, "STCS: Spatial–temporal collaborative sampling in flow-aware software defined networks," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 6, pp. 999–1013, Apr. 2020.

[3] M. Kamel, W. Hamouda, and A. Youssef, "Ultra-dense networks: A survey," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 4, pp. 2522–2545, 4th Quart., 2016.

[4] H. Guo, J. Liu, and J. Zhang, "Computation offloading for multi-access mobile edge computing in ultra-dense networks," *IEEE Commun. Mag.*, vol. 56, no. 8, pp. 14–19, Aug. 2018.

[5] H. Guo, J. Zhang, J. Liu, H. Zhang, and W. Sun, "Energy-efficient task offloading and transmit power allocation for ultra-dense edge computing," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2018, pp. 1–6.

[6] Y. Sun, S. Zhou, and J. Xu, "EMM: Energy-aware mobility management for mobile edge computing in ultra dense networks," *IEEE J. Sel. Areas Commun.*, vol. 35, no. 11, pp. 2637–2646, Nov. 2017.

[7] T. Ouyang, Z. Zhou, and X. Chen, "Follow me at the edge: Mobility-aware dynamic service placement for mobile edge computing," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 10, pp. 2333–2345, Sep. 2018.

[8] F. Rao and E. Bertino, "Privacy techniques for edge computing systems," *Proc. IEEE*, vol. 107, no. 8, pp. 1632–1654, Aug. 2019.

[9] X. Wang, C. Wang, X. Li, V. C. M. Leung, and T. Taleb, "Federated deep reinforcement learning for internet of things with decentralized cooperative edge caching," *IEEE Internet Things J.*, early access, Apr. 9, 2020, doi: 10.1109/JIOT.2020.2986803.

[10] R. Yang, F. R. Yu, P. Si, Z. Yang, and Y. Zhang, "Integrated blockchain and edge computing systems: A survey, some research issues and challenges," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 2, pp. 1508–1532, 2nd Quart., 2019.

[11] J. Konecný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon. (2016). *Federated Learning: Strategies for Improving Communication Efficiency*. [Online]. Available: http://arxiv.org/abs/1610.05492

[12] Q. Wu, K. He, and X. Chen, "Personalized federated learning for intelligent IoT applications: A cloud-edge based framework," *IEEE Open J. Comput. Soc.*, vol. 1, pp. 35–44, 2020.

[13] X. Wang, Y. Han, C. Wang, Q. Zhao, X. Chen, and M. Chen, "In-edge Ai: Intelligentizing mobile edge computing, caching and communication by federated learning," *IEEE Netw.*, vol. 33, no. 5, pp. 156–165, Jun. 2019.

[14] X. Wang *et al.*, "Convergence of edge computing and deep learning: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 2, pp. 869–904, 2nd Quart., 2020.

[15] E. Li, L. Zeng, Z. Zhou, and X. Chen, "Edge AI: On-demand accelerating deep neural network inference via edge computing," *IEEE Trans. Wireless Commun.*, vol. 19, no. 1, pp. 447–457, Oct. 2020.

[16] S. Luo, X. Chen, Q. Wu, Z. Zhou, and S. Yu, "HFEL: Joint edge association and resource allocation for cost-efficient hierarchical federated edge learning," *IEEE Trans. Wireless Commun.*, early access, Jun. 26, 2020, doi: 10.1109/TWC.2020.3003744.

[17] D. Liu, X. Chen, Z. Zhou, and Q. Ling, "Hiertrain: Fast hierarchical edge Ai learning with hybrid parallelism in mobile-edge-cloud computing," *IEEE Open J. Commun. Soc.*, vol. 1, pp. 634–645, 2020.

[18] X. Chen, Q. Shi, L. Yang, and J. Xu, "ThriftyEdge: Resource-efficient edge computing for intelligent IoT applications," *IEEE Netw.*, vol. 32, no. 1, pp. 61–65, Jan. 2018.

[19] S. Wang, Y.-C. Wu, M. Xia, R. Wang, and H. V. Poor, "Machine intelligence at the edge with learning centric power allocation," *IEEE Trans. Wireless Commun.*, early access, Jul. 28, 2020, doi: 10.1109/TWC.2020.3010522.

[20] C. Liu *et al.*, "A new deep learning-based food recognition system for dietary assessment on an edge computing service infrastructure," *IEEE Trans. Services Comput.*, vol. 11, no. 2, pp. 249–261, Jan. 2018.

[21] Y. He, N. Zhao, and H. Yin, "Integrated networking, caching, and computing for connected vehicles: A deep reinforcement learning approach," *IEEE Trans. Veh. Technol.*, vol. 67, no. 1, pp. 44–55, Jan. 2018.

[22] P. K. Sharma, M. Chen, and J. H. Park, "A software defined fog node based distributed blockchain cloud architecture for IoT," *IEEE Access*, vol. 6, pp. 115–124, 2018.

[23] Y. Huang, J. Zhang, J. Duan, B. Xiao, F. Ye, and Y. Yang, "Resource allocation and consensus on edge blockchain in pervasive edge computing environments," in *Proc. IEEE 39th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, 2019, pp. 1476–1486.

[24] (2012). *Distributed Computing, Storage and Radio Resource Allocation Over Cooperative Femtocells (TROPIC)*. [Online]. Available: http://www.ict-tropic.eu

[25] P. D. Lorenzo, S. Barbarossa, and S. Sardellitti, "Joint optimization of radio resources and code partitioning in mobile cloud computing," 2013. [Online]. Available: arXiv:1307.3835.

[26] D. Lopez-Perez, A. Valcarce, G. de la Roche, and J. Zhang, "OFDMA femtocells: A roadmap on interference avoidance," *IEEE Commun. Mag.*, vol. 47, no. 9, pp. 41–48, Sep. 2009.

[27] G. Ku and J. M. Walsh, "Resource allocation and link adaptation in LTE and LTE advanced: A tutorial," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 3, pp. 1605–1633, 3rd Quart., 2015.

[28] J. Kwak, Y. Kim, J. Lee, and S. Chong, "DREAM: Dynamic resource and task allocation for energy minimization in mobile cloud systems," *IEEE J. Sel. Areas Commun.*, vol. 33, no. 12, pp. 2510–2523, Dec. 2015.

[29] X. Zhou, M. Sun, G. Y. Li, and B. F. Juang, "Intelligent wireless communications enabled by cognitive radio and machine learning," *China Commun.*, vol. 15, no. 12, pp. 16–48, 2018.

[30] Y. Huang, C. Xu, C. Zhang, M. Hua, and Z. Zhang, "An overview of intelligent wireless communications using deep reinforcement learning," *J. Commun. Inf. Netw.*, vol. 4, no. 2, pp. 15–29, 2019.

[31] J. Jiang, S. Zhang, B. Li, and B. Li, "Maximized cellular traffic offloading via device-to-device content sharing," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 1, pp. 82–91, Jan. 2016.

[32] M. Chen, M. Dong, and B. Liang, "Resource sharing of a computing access point for multi-user mobile cloud offloading with delay constraints," *IEEE Trans. Mobile Comput.*, vol. 17, no. 12, pp. 2868–2881, Dec. 2018.

[33] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proc. IEEE*, vol. 107, no. 8, pp. 1738–1762, Aug. 2019.

[34] E. Cuervo *et al.*, "MAUI: Making smartphones last longer with code offload," in *Proc. ACM 8th Int. Conf. Mobile Syst. Appl. Services (MobiSys)*, Jun. 2010, pp. 49–62.

[35] S. E. Mahmoodi, R. N. Uma, and K. P. Subbalakshmi, "Optimal joint scheduling and cloud offloading for mobile applications," *IEEE Trans. Cloud Comput.*, vol. 7, no. 2, pp. 301–313, Apr. 2019.

[36] Y. Zhang, D. Niyato, and P. Wang, "Offloading in mobile cloudlet systems with intermittent connectivity," *IEEE Trans. Mobile Comput.*, vol. 14, no. 12, pp. 2516–2529, Feb. 2015.

[37] V. Smith, C. Chiang, M. Sanjabi, and A. S. Talwalkar, "Federated multi-task learning," in *Proc. Adv. Neural Inf. Process. Syst. 30th Annu. Conf. Neural Inf. Process. Syst.*, Dec. 2017, pp. 4424–4434. [Online]. Available: http://papers.nips.cc/paper/7029-federated-multi-task-learning

[38] Z. Shi, L. Li, Y. Xu, X. Li, W. Chen, and Z. Han, "Content caching policy for 5G network based on asynchronous advantage actor–critic method," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, 2019, pp. 1–6.

[39] MATLAB. (2020). *Reinforcement Learning Toolbox*. [Online]. Available: https://ww2.mathworks.cn/help/reinforcement-learning/index.html?s_tid=CRUX_lftnav

[40] S. T. ul Hassan, M. Bennis, P. H. J. Nardelli, and M. Latva-Aho, "Caching in wireless small cell networks: A storage-bandwidth tradeoff," *IEEE Wireless Commun. Lett.*, vol. 20, no. 6, pp. 1175–1178, Mar. 2016.

[41] Y. Jiang, J. Konecny, K. Rush, and S. Kannan, "Improving federated learning personalization via model agnostic meta learning," in *Proc. ICLR*, 2019, pp. 1–11.

Shuai Yu (Member, IEEE) received the B.S. degree from Nanjing University of Post and Telecommunications, Nanjing, China, in 2009, the M.S. degree from Beijing University of Post and Telecommunications, Beijing, China, in 2014, and the Ph.D. degree from the University Pierre and Marie Curie (currently, Sorbonne Université), Paris, France, in 2018.

He is currently a Postdoctoral Research Fellow with the School of Data and Computer Science, Sun Yat-sen University, Guangzhou, China. His research interests include wireless communications, mobile computing, and machine learning.

Xu Chen (Senior Member, IEEE) received the Ph.D. degree in information engineering from the Chinese University of Hong Kong, Hong Kong, in 2012.

He is a Full Professor with Sun Yat-sen University, Guangzhou, China, and the Vice Director of the National and Local Joint Engineering Laboratory of Digital Home Interactive Applications. He was a Postdoctoral Research Associate with Arizona State University, Tempe, AZ, USA, from 2012 to 2014, and a Humboldt Scholar Fellow with the Institute of Computer Science, University of Göettingen, Göettingen, Germany, from 2014 to 2016.

Prof. Chen was a recipient of the Prestigious Humboldt Research Fellowship awarded by Alexander von Humboldt Foundation of Germany, the 2014 Hong Kong Young Scientist Runner-Up Award, the 2017 IEEE Communication Society Asia–Pacific Outstanding Young Researcher Award, the 2017 IEEE ComSoc Young Professional Best Paper Award, the Honorable Mention Award of 2010 IEEE International Conference on Intelligence and Security Informatics, the Best Paper Runner-Up Award of 2014 IEEE International Conference on Computer Communications (INFOCOM), and the Best Paper Award of 2017 IEEE International Conference on Communications. He is currently an Area Editor of the IEEE OPEN JOURNAL OF THE COMMUNICATIONS SOCIETY and an Associate Editor of the IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS, the IEEE INTERNET OF THINGS JOURNAL, and the IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS Series on Network Softwarization and Enablers.

Zhi Zhou (Member, IEEE) received the B.S., M.E., and Ph.D. degrees from the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China, in 2012, 2014, and 2017, respectively.

He is currently an Associate Professor with the School of Data and Computer Science, Sun Yat-sen University, Guangzhou, China. He has been a Visiting Scholar with the University of Göttingen, Göttingen, Germany. His research interests include edge computing, cloud computing, and distributed systems.

Dr. Zhou was nominated for the 2019 CCF Outstanding Doctoral Dissertation Award. He is the sole recipient of the 2018 ACM Wuhan and Hubei Computer Society Doctoral Dissertation Award and a recipient of the Best Paper Award of IEEE UIC 2018.

Xiaowen Gong (Member, IEEE) received the B.Eng. degree in electronics and information engineering from Huazhong University of Science and Technology, Wuhan, China, in 2008, the M.Sc. degree in communications from the University of Alberta, Edmonton, AB, Canada, in 2010, and the Ph.D. degree in electrical engineering from Arizona State University, Tempe, AZ, USA, in 2015.

From 2015 to 2016, he was a Postdoctoral Researcher with the Department of Electrical and Computer Engineering, Ohio State University, Columbus, OH, USA. He is currently an Assistant Professor with the Department of Electrical and Computer Engineering, Auburn University, Auburn, AL, USA. His research interests are in the areas of wireless networks and mobile computing, with current focuses on edge computing, data crowdsourcing, and applications of ML/AI.

Dr. Gong received the IEEE INFOCOM 2014 Runner-Up Best Paper Award as a co-author and the ASU ECEE Palais Outstanding Doctoral Student Award in 2015.

Di Wu (Senior Member, IEEE) received the B.S. degree from the University of Science and Technology of China, Hefei, China, in 2000, the M.S. degree from the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China, in 2003, and the Ph.D. degree in computer science and engineering from the Chinese University of Hong Kong, Hong Kong, in 2007.

He was a Postdoctoral Researcher with the Department of Computer Science and Engineering, Polytechnic Institute of New York University, Brooklyn, NY, USA, from 2007 to 2009, advised by Prof. K. W. Ross. He is currently a Professor and the Associate Dean of the School of Data and Computer Science, Sun Yat-sen University, Guangzhou, China. His research interests include cloud/edge computing, multimedia communication, Internet measurement, and network security.

Prof. Wu was a co-recipient of the IEEE INFOCOM 2009 Best Paper Award and the IEEE Jack Neubauer Memorial Award in 2019.