Dynamic Service Caching Aided Computation Offloading Optimization Algorithm for Mobile-Edge Networks

Bo Xie, Jinhua Xie, Haixia Cui, *Senior Member, IEEE*, Yejun He, *Senior Member, IEEE*, and Mohsen Guizani, *Fellow, IEEE*

***Abstract*—The widespread adoption of computation- and communication-intensive applications, such as object detection, VR/AR, and telemedicine, has significantly alleviated transmission pressure on backbone networks and improved user experience. However, efficiently managing and computing these tasks on user sides remains a significant challenge, particularly under resource-constrained conditions. To address this problem, we propose a new service caching decision method based on deep dueling double Q-network (D3QN) by employing a learnable policy to handle the unknown task requests and determine the optimal caching strategies. Additionally, the limited storage capacity of edge servers (ES) is mitigated by forwarding the resource-intensive or infrequently requested tasks to the cloud data centers (CDC). The channel selection problem is modeled as a multiuser game and a distributed method is developed to achieve the Nash Equilibrium (NE). Simulation results demonstrate that the proposed method outperforms the existing benchmarks, showcasing its effectiveness in managing complex, dynamic environments.**

***Index Terms*—Channel selection, computation offloading, deep dueling double Q-network (D3QN), mobile cloud-edge cooperative computing, service caching.**

I. Introduction

THE DEPLOYMENT of computation- and communication-intensive tasks, such as object detection, virtual reality/augmented reality (VR/AR), and telemedicine, has been extensively adopted on user devices, driven by the continuous evolution of mobile internet technologies and the widespread proliferation of mobile devices [1]. These tasks exhibit distinct resource demands, such as, object detection relies heavily on GPUs and NPUs for computation, VR/AR requires both high computational power and communication bandwidth for real-time processing, and telemedicine emphasizes storage resources to manage large-scale medical data. Since these applications alleviate lots of transmission pressure on backbone networks and improve the user experience [2], [3], their diverse and dynamic resource requirements introduce significant challenges for computation offloading and resource coordination [4], [5], [6], [7], [8]. Despite the advancements in high-performance computing chips accelerating the computation ability, effectively managing these heterogeneous tasks remains a critical research focus in the field of computation offloading. In fact, the goal of managing large-scale tasks on user sides is to enhance the system energy efficiency and reduce the overall task execution latency [9], [10]. Although lots of tasks are deployed on user sides, some substantial data still needs to be transmitted, including the task-required data and task metadata (e.g., the execution environment dependencies). This allows the users to send requests to service without the need to reupload task metadata to edge server (ES). A straightforward solution involves adopting a CDN-like concept by caching task metadata on user-side servers, referred to as ESs, a mechanism known as service caching. However, executing large-scale tasks across multiple ESs often results in competition for computational resources, significantly degrading task execution efficiency and leading to low resource utilization. Such inefficiency further exacerbates the imbalance in resource usage. Consequently, determining when to cache data and how to coordinate tasks has emerged as a critical focus in the field of computation offloading. Current research employs a cloud-edge-device architecture to efficiently orchestrate tasks, allocating computationally intensive tasks to appropriate layers and, within those layers, to optimal computing nodes.

The decision-making processes for computation offloading and service caching within the three-layer cloud-edge-device architecture are inherently nonlinear and significantly influenced by the dynamically changing system environment [14]. The nonlinearity of these processes, combined with the dynamic nature of the system environment, results in high computational complexity and limited generalization performance for traditional optimization-based

Received 18 November 2024; revised 25 January 2025 and 6 March 2025; accepted 26 March 2025. Date of publication 31 March 2025; date of current version 27 June 2025. This work was supported in part by the National Key Research and Development Program of China under Grant 2023YFE0107900; in part by the GuangDong Basic and Applied Basic Research Foundation under Grant 2024A1515012052; and in part by the National Natural Science Foundation of China under Grant 61871433, Grant 61828103, Grant 61201255, and Grant 62071306. *(Bo Xie and Jinhua Xie contributed equally to this work.) (Corresponding author: Haixia Cui.)*

Bo Xie, Jinhua Xie, and Haixia Cui are with the School of Electronic Science and Engineering (School of Microelectronics), South China Normal University, Foshan 528225, China (e-mail: bo.xie@m.scnu.edu.cn; 2022024840@m.scnu.edu.cn; cuihaixia@m.scnu.edu.cn).

Yejun He is with the College of Electronics and Information Engineering, Shenzhen University, Shenzhen 518060, China (e-mail: yjhe@szu.edu.cn).

Mohsen Guizani is with the Machine Learning Department, Mohamed Bin Zayed University of Artificial Intelligence, Abu Dhabi, UAE (e-mail: mguizani@ieee.org).

Digital Object Identifier 10.1109/JIOT.2025.3555978

approaches. Consequently, deep-reinforcement-learning (DRL) has garnered attention in the field of computation offloading due to its capability to make efficient decisions in complex and dynamic environments while exhibiting a degree of robustness [17]. DRL serves to control the selection of computing nodes for service caching and to determine service update actions. Additionally, considering the multichannel nature of communication links and the interference between channels, selecting an appropriate channel for data transmission is critically important. Therefore, under resource-constrained conditions, there exists a coupling relationship between computation offloading, service caching, and data transmission, making the realization of collaborative optimization from a global perspective a key challenge.

In this article, we propose a service caching decision method based on the deep dueling double Q-network (D3QN) by utilizing a learnable policy to handle the unknown user task requests and determine the optimal service caching strategy. Additionally, the ES forwards the resource-intensive or infrequently requested tasks to CDC with larger storage capacity and in fact it has limited storage resources. So, we apply D3QN to address the joint optimization of service caching and channel selection in the dynamic edge environments. The formulated problem involves interdependent decisions and adaptability to the changing system conditions. We model the channel selection problem as a multiuser game and develop a distributed method to solve the Nash Equilibrium (NE). Unlike the traditional approaches which rely heavily on the fixed heuristics or static assumptions, our method integrates service caching and channel selection decisions (CSD) within a scalable and real-time framework. Simulation experiments demonstrate that the D3QN method outperforms the benchmark approaches, highlighting its potential and effectiveness.

The principal contributions of this article are summarized as follows.

1) We propose a novel D3QN-based service caching decision method that dynamically handles the user task requests and optimizes the caching strategies in a resource-constrained multitier architecture.
2) A multiuser channel selection algorithm is developed by the NE theory to address the data transmission challenge under interference conditions.
3) The experiments validate the efficiency and robustness of our approach and showcase significant improvements in energy efficiency and latency reduction compared to the existing methods.
4) Our proposed method highlights the scalability and adaptability to enable the effective resource management in dynamically evolving environments.

This article is structured as follows. Section II summarizes the related work. Section III describes the MEC network architecture and the process of task offloading. Section IV provides a detailed explanation of the proposed method. Section V discusses the experimental analysis and results. Section VI concludes this article.

II. Related Work

In this section, we examine the related research on joint computation offloading and service caching, categorizing existing work into two main approaches: 1) traditional optimization algorithms and 2) DRL methods.

A. Conventional Optimization Methods

A number of studies employ multiobjective or heuristic algorithms to tackle the coupled problems of service caching and computation offloading. In [15], a many-objective evolutionary approach (enhanced NSGA-III) is developed to jointly optimize computation offloading and service caching in MEC. Although this method effectively addresses dimensional explosion in multiobjective scenarios, it can be slow for large-scale real-world deployments. Cheng et al. [16] adopted a dynamic programming-based caching algorithm combined with a matching strategy for task offloading in vehicular-edge computing (VEC) networks, but the overall complexity remains high.

Cooperative and community-based frameworks also appear in the literature. For example, Liao et al. [12] introduced a two-stage approach wherein base stations are clustered into communities based on service-type similarity before a cost-based service caching and offloading algorithm is applied. Container-based systems receive special attention in [19], which formulates a nonlinear integer programming (NLIP) model to achieve globally optimal offloading and caching decisions, explicitly considering container startup times. Similarly, [21] leverages physical-layer full-duplex capabilities to jointly optimize caching and offloading, highlighting interference management and resource allocation in full-duplex-enabled MEC networks.

Other works target domain-specific extensions. In UAV-assisted scenarios, [22] combines trajectory planning, offloading, caching, and migration into a single framework using classical optimization techniques, whereas [24] proposes a two-layer optimization algorithm for secure offloading and caching in reconfigurable-intelligent-surface (RIS)-assisted networks. Mobility and coverage concerns are tackled in [33], which employs a genetic algorithm to place and update caches based on user location and orientation, and [37] addresses secure offloading and caching through a nonconvex problem formulation in open wireless channels. Further examples include [38], which optimize offloading and caching for UAV-based systems by incorporating user preferences and security constraints. Lastly, [39] proposes a vehicle-assisted cooperative caching system (VaCo), using multiswarm collaborative optimization to minimize service failure rates and costs in highly dynamic VEC.

These conventional approaches successfully reduce energy consumption, latency, or both by leveraging analytical modeling, multiobjective evolutionary algorithms, and domain-specific optimizations. However, their reliance on comprehensive system knowledge and often high computational complexity can limit real-time adaptability in large-scale or fast-changing edge environments [28].

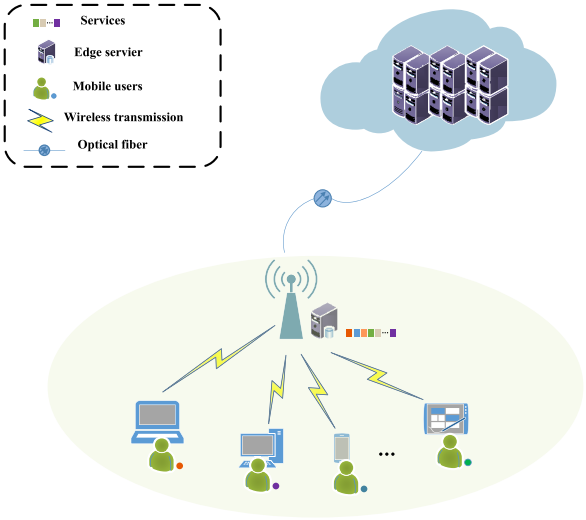

Fig. 1. System model.

B. Deep-Reinforcement-Learning-Methods

DRL has been widely adopted to address complex optimization problems in dynamic and nonlinear environments, particularly in the context of computation offloading and resource allocation. For instance, [18] employs dueling deep Q-networks (DDQN) to explore large action spaces and improve computational efficiency, while [26] integrates a meta-reinforcement learning (meta-RL) framework with DRL, introducing the MR-DRO algorithm for efficient offloading decisions. In [20], a deep Q-network (DQN) approach is used to minimize computational costs, and [27] leverages DQN to predict task popularity, dynamically adjusting service caching and computation offloading decisions to reduce system latency. Dual DQNs are employed in [23] for sequential subtask offloading in mobile vehicular environments, whereas [25] models server selection as a Markov decision process via DRL. Beyond focusing solely on either computation offloading or resource allocation, recent research increasingly explores joint optimization of service caching and offloading in MEC. For example, the LSTM-DDPG-based offloading strategy combined with a distributed hash table (DHT) for edge service caching was proposed in [28], effectively controlling algorithmic complexity despite the enlarged action space. Similarly, the dual-timescale DTTD3 scheme for multiUAV-assisted MEC systems was introduced in [29], updating service caching at a larger timescale while managing offloading and resource allocation at a smaller timescale to balance latency and energy consumption. A dynamic caching approach based on the 0–1 knapsack model was employed in [30] to handle directed acyclic graph (DAG) tasks, wherein DDPG is used for offloading decisions, facilitating efficient coordination among ESs with limited cache capacity. Meanwhile, the authors presented a hierarchical DRL framework that splits the service caching and computation offloading decisions between multiple edge agents (using DQN) and a cloud agent (using SAC) in [31], thereby adapting to the multilayered edge–cloud environment. Finally, the authors combined DRL with convex optimization techniques to form a hybrid soft actor–critic (SAC) scheme in [32], separately optimizing service caching and computation offloading, thus maintaining strong convergence and accuracy in high-dimensional action spaces.

From the above, DRL-based methods have significantly enhanced the decision-making in both service caching and computation offloading by improving convergence and adaptability. However, some challenges like Q-value overestimation and achieving rapid convergence in dynamic settings still need to be addressed. Existing approaches to computation offloading and service caching often encounter scalability and adaptability challenges in rapidly changing environments. To address this gap, we propose a D3QN-based framework that integrates service caching with a channel selection mechanism inspired by the interference-aware strategies presented in [21] which although not originally designed for channel selection, effectively accounts for intersignal interference—to manage multiuser competition and dynamic system fluctuations.

III. System Model

In this section, we first present the system model of MEC. Subsequently, the channel selection and service caching model are outlined with a focus on the QoE model while considering the energy consumption constrained by the service latency.

As depicted in Fig. 1, we tackle the optimization problem of channel selection and service caching within the cloud-edge-end architecture, which involves MUs, an ES, and a CDC. The communication between MUs and ES occurs via a wireless channel through a tiny base station, while the communication between ES and CDC takes place through a wired channel provided by optical fibers. The service repository is deployed on the CDC, and the ES downloads services from this repository to update the ES's cache space. When the MUs offload tasks to the ES or CDC, they only need to transmit the data required for processing, as the ES or CDC already has the necessary services for those tasks. Due to the limited cache space available at the ES, which cannot accommodate all services, some of the computational tasks are forwarded to CDC with larger cache capacities for processing. Similar to [33] and [34], we assume that each MU's current task must be completed within the current time slot.

The MUs are denoted as $\mathcal{U} = \{1, 2, \ldots, u, \ldots, U\}$, and the tasks are represented as $\mathcal{K} = \{1, 2, \ldots, k, \ldots, K\}$. Additionally, the system operates over T fixed-length time slots, which is defined as $\mathcal{T} = \{1, 2, \ldots, t, \ldots, T\}$. It should be noted that we assume each MU selects a task from the set $\mathcal{K}$ to determine whether it should be offloaded. The task offloading decision (TOD) vector of MU u at time slot t is defined as $\xi_u(t) \in \{0, 1\}$, where $\xi_u(t) = 0$ indicates the task is executed locally and $\xi_u(t) = 1$ indicates the task is offloaded to the ES. Note that if there is no corresponding service on ES, the task will be transferred to CDC for further processing.

Each task $k \in \mathcal{K}$ can be represented as (D_k, I_k, S_k), where D_k denotes the memory size occupied by the corresponding service of task k, I_k represents the input size of task k, and S_k indicates the computational power required to execute task k (i.e., the CPU cycles needed). Let $\beta_k^{(t)} \in \{0, 1\}$ denote whether the service for task k is cached, referred to as the service cache decision (SCD). When $\beta_k^{(t)} = 0$, it signifies that the service is not cached; when $\beta_k^{(t)} = 1$, it means that the service is cached. The memory size of the ES is defined as C, and its caching capacity satisfies

$$\sum_{k \in K} D_k \beta_k^{(t)} \leq C \quad \forall t \in T. \tag{1}$$

The updating services decision (USD) on ES at time slot t is denoted as $\rho_k^{(t)} \in \{-1, 0, 1\}$, where $\rho_k^{(t)} = -1$ and $\beta_k^{(t)} = 1$ indicate the removal of the service from the ES at time $t+1$, $\rho_k^{(t)} = 0$ and $\beta_k^{(t)} = 1$ indicate that the service will continue to be cached, $\rho_k^{(t)} = 0$ and $\beta_k^{(t)} = 0$ indicate that the service is not cached by the ES, and $\rho_k^{(t)} = 1$ and $\beta_k^{(t)} = 0$ indicate that the service will be cached by the ES at time $t+1$.

A. Network Model

In the proposed three-tier framework, each MU has communication with the ES via a wireless channel, and the ES has communication with the CDC through a wired channel facilitated by optical fibers.

1) Communication Between MU and ES: We assume the total available bandwidth from the MU to ES is evenly divided into N orthogonal wireless channels, $\mathcal{N} = \{1, 2, \ldots, n, \ldots, N\}$, with each MU restricted to using a single channel to interact with the ES. This orthogonal channel allocation strategy ensures that there is no interference between users utilizing different orthogonal channels within the same time slot. In this article, the CSD of all users in time slot t are denoted by $\eta_t = \{\eta_{1,t}, \eta_{2,t}, \ldots, \eta_{u,t}, \ldots, \eta_{U,t}\}$, where $\eta_{u,t} = 0$ indicates that MU u completes the task locally. $\eta_{u,t} = n \in \mathcal{N}$ indicates that the MU u chooses the channel n to offload its task to the ES. The code division multiple access (CDMA) technology allows multiple mobile users to transmit data simultaneously on the same spectrum resources. According to this model, the uplink transmission rate of mobile user u during time slot t is defined by Shannon's formula

$$r_{u,e}^t = W_u^n \log_2 \left(1 + \frac{q_u g_u^n}{\delta^2 + \sum_{v \in U/\{u\}} q_v g_v^e}\right) \tag{2}$$

where W_u^n is the channel bandwidth between the MU u and the ES, δ^2 denotes the power of the background noise, q_u is the transmission power of the MU u, and g_u^e is the transmission gain of MU u, which depends on the distance between the ES and the MU.

2) Communication Between ES and CDC: The ES communicates with CDC via fiber optics, and the transmission rate in time slot t is denoted by $r_{e,c}^t$.

B. Computation Model

The system allows tasks to be executed locally, on ES, or on CDC. Only when a corresponding service is cached on ES can a task be executed on ES; otherwise, it will be offloaded to CDC. Similar to previous studies [25], [33], we do not consider the overhead of MUs checking for cached services at the ES, as this overhead is minimal.

1) Local Computation: When MU u performs task k locally, the computational capability (CPU cycles per second) of MU u is denoted as $f_{\ell,k}$. The task must be completed within the current time slot, therefore the CPU frequency must satisfy $f_{\ell,k} \geq S_k/\tau$, where τ represents the duration of the time slot. According to [36], the energy consumption increases as the frequency of MU u squared. Hence, the energy consumption of MU u carrying out task k can be expressed as

$$E_{u,\ell}^t = \zeta \cdot \left(f_{\ell,k}\right)^2 S_k \tag{3}$$

where ζ is a coefficient.

2) Cloud Offloading: If MU U offloads task k to the ES and the ES does not have the service for task k cached, the inputs of task k are uploaded through ES to the CDC. According to [14], the ES is required to download the service from the repository at each time slot t. However, obtaining services can be time-consuming, especially during peak times. Therefore, similar to [11] and [13], we prohibit the ES from remotely accessing the repository without first caching the services; instead, it directly sends tasks from ES to CDC. Let $f_c (f_c \gg f_{\ell,k} \quad \forall k \in \mathcal{K})$ represent the computational capacity of CDC. The task delay on CDC can be represented as

$$D_{u,c}^t = \frac{S_k}{f_c} + \frac{I_k}{r_{u,e}^t} + \frac{I_k}{r_{e,c}^t}. \tag{4}$$

The first part pertains to the task execution delay, while the second and third parts represent the data transmission delay. In order to ensure completion within the current time slot t, it is necessary for the delay to satisfy $D_{u,c}^t \leq \tau$. The transmission energy consumption of MU u transmitting task k can be expressed as

$$E_{u,c}^t = p_k \left(\frac{I_k}{r_{u,e}^t} + \frac{I_k}{r_{e,c}^t}\right) \tag{5}$$

where p_k refers to the transmission power. Due to the reliable power supply of CDC, we do not take into account the energy consumption of computing tasks in these facilities.

3) Edge Offloading: When MU offloads task k to the ES for execution in time slot t, and the ES has already cached the service for task k, only the input data of task k needs to be uploaded, and then the ES can directly process task k. Therefore, the execution delay can be represented as

$$D_{u,e}^t = \frac{S_k}{f_e} + \frac{I_k}{r_{u,e}^t} \tag{6}$$

where f_e is the computing capacity of the ES. The execution delay $D_{u,e}^t$ should also satisfy $D_{u,e}^t \leq \tau$.

Additionally, the transmission energy consumption is given by

$$E_{u,e}^{t} = p_{k,\text{trans}} \frac{I_k}{r_{u,e}^{t}} + p_{k,exe} \frac{S_k}{f_e} \tag{7}$$

where $p_{k,\text{trans}}$ refers to the transmission power, while $p_{k,exe}$ represents the execution power.

C. Problem Formulation

Our goal is to minimize the energy usage of MUs while ensuring that task latency constraints are met. According to the computation offloading model described above, the energy consumption of MUs at the time slot t can be expressed as

$$E_t = \sum_{u\in\mathcal{U},k\in\mathcal{K}} \begin{cases} E_{u,l}^{t} \text{ if } \eta_{u,t}=0 \text{ and } \xi_u^{(t)}=0 \\ E_{u,e}^{t} \text{ if } \eta_{u,t}\neq 0 \text{ and } \xi_u^{(t)}=k,\ \beta_k^{(t)}=1 \\ E_{u,c}^{t} \text{ if } \eta_{u,t}\neq 0 \text{ and } \xi_u^{(t)}=k,\ \beta_k^{(t)}=0 \end{cases} \tag{8}$$

where $\xi_u^{(t)}=0$ indicates the task is executed locally, resulting in no need for transmission to a remote location and thus $\eta_{u,t}=0$. If the task is offloaded to the ES, the user decides to offload the task to the ES ($\xi_u^{(t)}=k$), selects a channel to transmit data ($\eta_{u,t}\neq 0$), and checks if the ES has cached the service corresponding to the task ($\beta_k^{(t)}=1$). In cases where the ES has not cached the service corresponding to the task ($\beta_k^{(t)}=0$), it further offloads the task to CDC, requiring selection of a channel for data transmission as well ($\eta_{u,t}\neq 0$). So, the opyimization problem can be formulated as

$$\min_{\eta_t,\beta_t} \lim_{T\to\infty} \frac{1}{T}\sum_{t=1}^{T} E_t \tag{9}$$

$$\text{s.t.} \quad \sum_{k\in\mathcal{K}} D_k \beta_k^{(t)} \le C \quad \forall t\in\mathcal{T} \tag{10}$$

$$\beta_k^{(t+1)} = \beta_k^{(t)} + \rho_k^{(t)} \quad \forall t\in\mathcal{T}\ \forall k\in\mathcal{K} \tag{11}$$

$$\rho_k^{(t)} + \beta_k^{(t)} \ge 0 \quad \forall t\in\mathcal{T}\ \forall k\in\mathcal{K} \tag{12}$$

$$D_{u,e}^{t} \le \tau \quad \forall k\in\mathcal{K}\ \forall u\in\mathcal{U} \tag{13}$$

$$D_{u,c}^{t} \le \tau \quad \forall k\in\mathcal{K}\ \forall u\in\mathcal{U} \tag{14}$$

$$\rho_k^{(t)} \in \{-1,0,1\} \quad \forall t\in\mathcal{T}\ \forall k\in\mathcal{K} \tag{15}$$

$$\eta_{u,t} \in \{0,1,\ldots,N\} \quad \forall u\in\mathcal{U}\ \forall t\in\mathcal{T} \tag{16}$$

$$\beta_k^{(t)} = \{0,1\} \quad \forall k\in\mathcal{K} \tag{17}$$

where (10) represents the restriction on the memory capacity of ES. Constraint (11) delineates the SCD. Constraint (12) specifies that the ES is prohibited from removing services that have not been previously cached. Constraints (13) and (14) ensure that the task delay does not exceed the duration of the time interval. Constraint (15) is the USD. Constraints (16) and (17) address the computational options for tasks, where $\eta_{u,t}=0$ indicating the task k execution locally, $\eta_{u,t}=n$ and $\beta_u^{(t)}=1$ suggest that the task k is offloaded to the ES through the channel n, and $\eta_{u,t}=n$ and $\beta_u^{(t)}=0$ indicating that the task k is offloaded to the CDC due to insufficient service caching of ES.

IV. Solution Methods

To solve the problem in (9), we decompose the original problem into two smaller subproblems: optimizing the service cache strategy and improving the channel selection scheme. Specifically, we first established a service caching policy using an MDP model, and then solved the problem using the DQN algorithm. To address the issue of overestimating Q-values and improving learning efficiency, we introduced the D3QN algorithm. Finally, we modeled the channel selection problem as a multiuser game and developed a distributed algorithm to determine the NE solution.

A. D3QN

The double DQN cannot accurately use the state value V and advantage value $\hat{A}$, leading to inaccurate results [35]. To solve this problem, the D3QN algorithm reconfigures the network structure by dividing the output layer into two parts: one for calculating the state value function V, and the other for calculating the action advantage function $\hat{A}$. Therefore, the formula for Q value is as follows:

$$q(s_t,a_t,\theta_t) = V(s_t,\theta_t) + \left(\hat{A}(s_t,a_t,\theta_t) - \frac{1}{|\mathcal{A}|}\sum_{a_t'\in\mathcal{A}} \hat{A}(s_t,a_t',\theta_t)\right) \tag{18}$$

where the action space $\mathcal{A}$, with size $|\mathcal{A}|$, and the state at time slot t denoted as s_t, along with the parameters of the neural network represented by θ_t. The state value function is given by $V(s_t,\theta_t)$, while the advantage value function is denoted as $\hat{A}(s_t,a_t,\theta_t)$. However, despite employing DDQN, which still utilizes the DQN method for updating state values, it suffers from the same bias maximization issue as DQN and may potentially lead to overestimation of state values.

D3QN uses a dual network architecture to estimate state values. This includes an evaluation network and a target network, with the evaluation network responsible for selecting the next action, and the target network addressing the overestimation problem by updating the parameter θ_t as shown in Fig. 2. Additionally, D3QN incorporates the DDQN's network structure between the last hidden layer and output layer, introducing the state value function $V(s_t,\theta_t)$ and advantage value function $\hat{A}(s_t,a_t,\theta_t)$. Here, $V(s_t,\theta_t)$ is dependent on state s_t, whereas $\hat{A}(s_t,a_t,\theta_t)$ is influenced by both state and action. This approach enables D3QN to more accurately evaluate each action and provide precise state value.

B. Service Cache Update Algorithm Based on D3QN

The key to solving the problem in (9) lies in obtaining the CSD η_t and the optimal USD ρ_t. We decompose the objective (9) in time slot t as follows:

$$\min_{\rho_t} \lim_{T\to\infty} \frac{1}{T}\sum_{t=1}^{T} E_t \tag{19}$$

$$\text{s.t. (10)}-\text{(14)}. \tag{20}$$

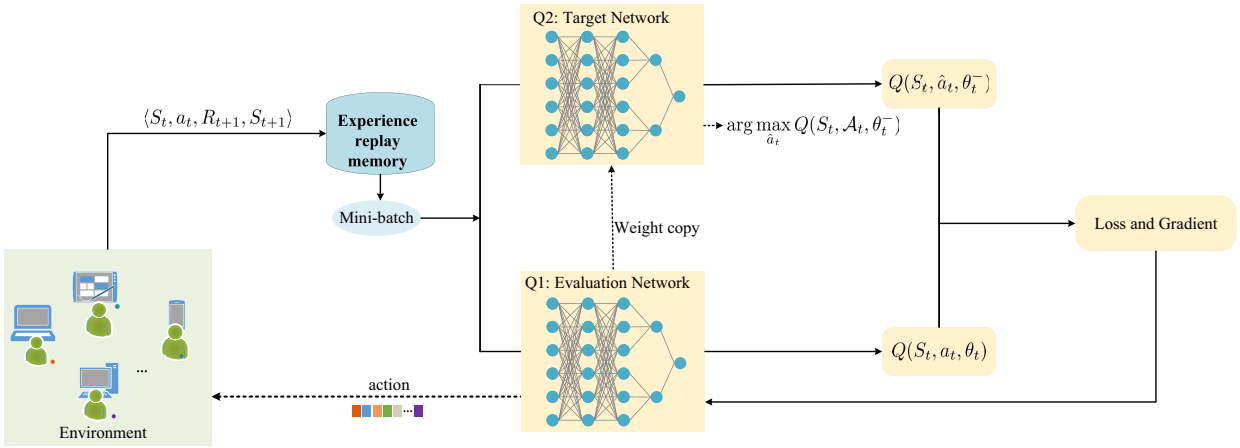

Fig. 2. Structure of D3QN.

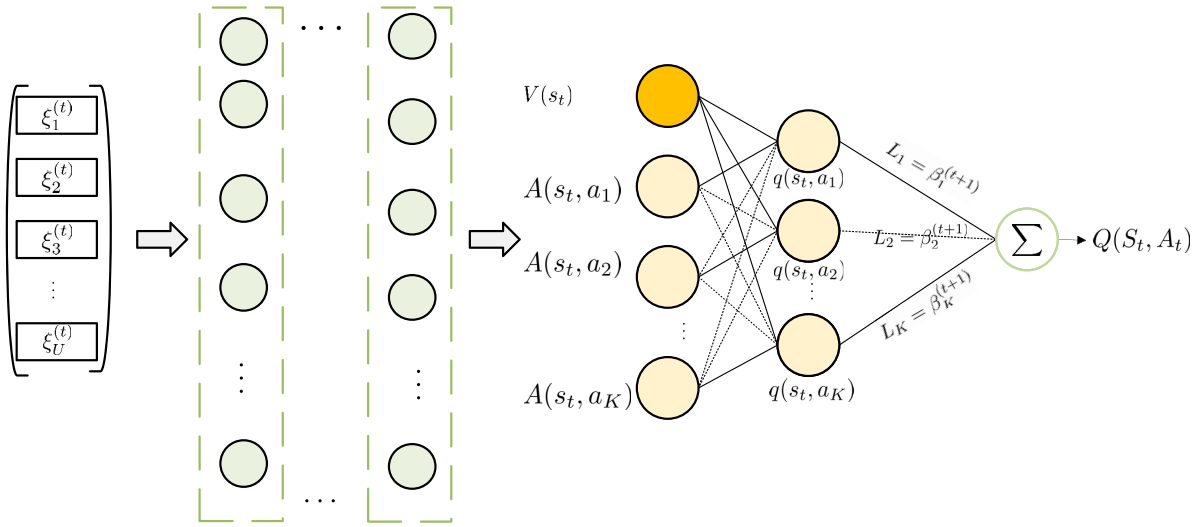

Fig. 3. Architecture of the proposed AOF-DNN.

In order to facilitate the resolution of problem (19), we first determine the optimal SCD β_{t+1} for time slot $t+1$. Subsequently, we derive the USD ρ_t for time slot t based on the equation $\beta_k^{(t+1)} = \rho_k^{(t)} + \beta_k^{(t)}$. The formulation of the optimal service caching state problem can be written as

$$\min_{\beta_{t+1}} \lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} E_t \tag{21}$$

$$\text{s.t.} \quad \sum_{k\in\mathcal{K}} D_k \beta_k^{(t)} \le C \quad \forall t \in \mathcal{T} \tag{22}$$

$$\beta_k^{(t+1)} \in \{0, 1\} \quad \forall k \in \mathcal{K}. \tag{23}$$

We convert the problem (21) into a Markov decision process, where ES acts as the agent, and the state space, action space, and reward function can be summarized as follows:

1) State Space: The state s_t at time slot t is given as $\xi_t = \{\xi_1^{(t)}, \xi_2^{(t)}, \ldots, \xi_u^{(t)}, \ldots, \xi_U^{(t)}\}$. In this context, $\xi_u^{(t)} = 0$ indicates that the MU u has no request, while $\xi_u^{(t)} = k$ signifies that the MU u requests to execute task k. Note that we utilize random functions to simulate ξ in this article.

2) Action Space: The action a_t at time slot t represents the caching decision for services, denoted as $a_t = \beta_{t+1} (\beta_{t+1} \in \{0, 1\})$. Each bit in β_{t+1} indicates whether a specific service will be cached on the ES in the next time step $t+1$: $\beta_{t+1} = 1$ signifies that the corresponding service will be cached, while $\beta_{t+1} = 0$ signifies that the service will not be cached.

3) Reward Function: We strive to minimize energy usage, which conflicts with the objective of maximizing cumulative discounted rewards in DRL. As a result, the value of the objective function should be inversely related to the reward value. The agent is rewarded based on its current state and corresponding action at time slot t. In order to reduce transmission energy consumption for task offloading, the immediate reward is defined as $r_{t+1} = -E_{t+1}$.

The objective of D3QN is to acquire the MU request ξ and forecast the optimal SCD for the next time slot based on user requests ξ in the current time slot. To enhance the accuracy of state value, we decompose the calculation of the Q-value into two components: the state value $V(s_t, \theta_t)$ and the advantage value $\hat{A}(s_t, a_t, \theta_t)$. Additionally, to address high-dimensional

action space resulting from service heterogeneity and improve the learning efficiency of D3QN, we introduce a novel DNN design known as action output fusion (AOF). The AOF architecture dynamically integrates caching operations using a two-layer structure at the output layer, as depicted in Fig. 3. In AOF-DNN, MUs' task requests ξ serve as input to the DNN, with both state value $V(s)$ and advantage value $V(s, a)$ incorporated between the last hidden layer and output layer. The computation of Q-value for each action is outlined in (18), enabling derivation of each action's value through state values and advantage values via the D3QN algorithm for more precise results. Moreover, we utilize a two-layer architecture (TLA) as the output layer of the AOF-DNN to expedite the convergence of D3QN, as shown in Fig. 3. The first layer of TLA comprises K neurons, with each neuron corresponding to task K. Denoting the output of the first layer as $q_t = (q(s_t, a_1), q(s_t, a_2), \ldots, q(s_t, a_K))$, where $q(s_t, a_K)$ represents the partial state-action value for task k in service caching. The second layer of TLA consists of only one neuron without an activation unit, which outputs the weighted sum of all (s_t, a_k). We represent the weights connecting the first and second layers in the output layer as $L = (L_1, L_2, \ldots, L_K)$, where L_K is specifically associated with connecting the k-th unit in the first layer to the second layer. To identify $q(s_t, a_k)$ for a specific action a_k, we assign the value of a_k to L_k, i.e., $L_k = \beta_k^{(t+1)} (k \in K, \beta_k^{(t+1)} \in \{0, 1\})$. Subsequently, the DNN computes this weighted sum of $q(s_t, a_k)$ to produce Q-value for action sequence A_t, i.e., $Q(S_t, A_t) = \sum_{i=1}^{U} \beta_i^{(t+1)} q(s_t, a_i)$.

As illustrated in Fig. 2, the ES employs a δ-greedy strategy to cache services during the training phase at the end of time slot t, where the probability of caching services randomly is $1 - \delta$, and the probability of caching services based on $A_t^* = \arg\max_a Q(S_t, a_t, \theta_t)$ is $1 - \delta$.

When the task requests ξ_t of MUs are input into the evaluation network, the AOF-DNN produces the predicted state-action value using a greedy strategy, denoted as $Q(S_t, A_t, \theta_t)$. The expected state-action value is then derived using the target network according to the specific formula

$$\bar{Q}\left(S_t, A_t, \theta_t^-\right) = R_{t+1} + \gamma Q\left(S_t, \arg\max_a Q\left(S_t, A_t, \theta_t^-\right), \theta_t^-\right) \tag{24}$$

where $\gamma \in \{0, 1\}$ is the discount factor, θ_t^- is the network parameter of the target network, and $\arg\max_a Q(S_t, A_t, \theta_t^-)$ is provided by the target network.

During the D3QN inference phase, ξ_t is fed into the DNN, and the initial layer of the AOF-DNN output layer generates $q_t = (q(s_t, a_1), q(s_t, a_2), \ldots, q(s_t, a_K))$. As each a_k consumes a specific amount of memory in ES and ES's memory is limited, determining the optimal caching state at t+1 involves finding $\max_{\beta_{t+1}} \sum_{i=1}^{U} \beta_i^{(t+1)} q(s_t, a_i)$. This optimal caching state problem can be expressed as follows:

$$\max_{\beta_{t+1}} \sum_{i=1}^{U} \beta_i^{(t+1)} q(s_t, a_i) \tag{25}$$

$$\text{s.t.} \quad \sum_{k \in \mathcal{K}} D_k \beta_k^{(t+1)} \le C \quad \forall t \in \mathcal{T} \tag{26}$$

$$\beta_k^{(t+1)} \in \{0, 1\}. \tag{27}$$

The problem described in (25) is a typical knapsack problem as discussed in [40]. We employ a dynamic programming approach to determine the optimal solution. To do thus, we introduce a $K \times C$ matrix ϕ, where $\phi_v(K, C)$ records the maximum caching value under different capacities and $\phi_r(K, C)$ records the decision on whether to cache the corresponding service.

The detailed computational complexity analysis for the above D3QN algorithm can be found in the Appendix.

C. Multiuser Channel Selection Algorithm

Up to this point, we have been able to identify the optimal service caching decision β_t^* for all users within any time slot, given any CSD η_t and user TOD ξ_t. By replacing β_t^* in the problem (9), it can be converted into a multiuser channel selection problem

$$\min_{\eta_t} \Gamma_t(\eta_t) = \sum_{k=1}^{K} E_t \tag{28}$$

$$\text{s.t.} \quad (16) \text{ and } (17). \tag{29}$$

where $\eta_t = \{\eta_{1,t}, \eta_{2,t}, \ldots, \eta_{U,t}\}$, $\eta_{u,t} (u \in U)$ has $N + 1$ choice values. Addressing problem (28) poses challenges due to its reliance on combinatorial optimization within a multidimensional discrete space $\{0, 1, 2, \ldots, N\}$.

In this game, the set of MUs $\mathcal{U}$ represents the player community in the game, and each MU u has its own strategy space $\Psi_{u,t}$ to make TODs. The goal of MU u is to minimize its energy consumption $\Gamma_t(\eta_t)$ at time step t.

Specifically, the multiuser game G is given by

$$G = \langle \mathcal{U}, \{\Psi_{u,t}\}_{u \in \mathcal{U}}, \Gamma_t(\eta_t) \rangle. \tag{30}$$

In this game, each MU u independently chooses its optimal CSD $\eta_{u,t}$, aiming to achieve the minimum energy consumption. Let $\eta_{u^-,t} = \{\eta_{1,t}, \eta_{2,t}, .., \eta_{u-1,t}, \eta_{u+1,t}, \ldots, \eta_{U,t}\}$ represents the CSD of all other users except MU u. The MU u is able to independently select its optimal CSD η_t^* in order to minimize energy consumption, given $\eta_{u^-,t}$

$$\eta_{u,t}^* = \arg\min_{\eta_{u,t}} \Gamma_t\left(\eta_{u^-,t}, \eta_{u,t}\right). \tag{31}$$

Therefore, the goal of the multiuser game is to find the NE, that is $\eta_t^* = \{\eta_{1,t}^*, \eta_{2,t}^*, \ldots, \eta_{u,t}^*, \ldots, \eta_{U,t}^*\}$. In other words, at time slot t, no user can further reduce their computational cost by adjusting their decisions. In other words, $\Gamma_t(\eta_{u,t}^*, \eta_{u^-,t}^*) \le \Gamma_t(\eta_{u,t}, \eta_{u^-,t}^*) \quad \forall u \in \mathcal{U}, \quad \eta_{u,t} \in \Psi_{u,t}$, thus achieving the NE solution. In this game, if the cost of computing the task locally is greater than the cost of offloading the task to ES, MU u would be more inclined to choose offloading the task to ES. Only when ES does not have the corresponding service will ES send the task to CDC for processing

$$\left(1 - \beta_k^{(t)}\right) E_{u,c,k}^t + \beta_k^{(t)} E_{u,e,k}^t \le E_{u,\ell,k}^t. \tag{32}$$

By substituting formulas (3), (5) and (7) into (32), we derive

$$p_k \frac{I_k}{r_{u,e}^t} + \left(1 - \beta_k^{(t)}\right) p_k \frac{I_k}{r_{e,c}^t} \leq \varsigma \left(f_{\ell,k}\right)^2 S_k \tag{33}$$

and

$$\varsigma \left(f_{\ell,k}\right)^2 S_k - \left(1 - \beta_k^{(t)}\right) p_k \frac{I_k}{r_{e,c}^t} \geq 0. \tag{34}$$

That is, the energy consumption of the task when executed locally is greater than the energy consumption of the task during network transmission between the ES and CDC. To ensure that each user can efficiently make CSD, we define an interference threshold $\gamma_{k,t} = \sum_{v \in \mathcal{U} \setminus \{u\}} q_v g_v^e$, which satisfies the following inequality

$$\begin{aligned} \gamma_{k,t} &= \sum_{v \in \mathcal{U} \setminus \{u\}, \eta_{u,t} = \eta_{v,t}} q_v g_v^e \\ &\leq \frac{q_v g_v^e}{I_k} \left(\frac{q_v g_v^e}{2W_u^e \left(\varsigma (f_{\ell,k})^2 S_k - (1 - \beta_k^{(t)}) p_k \frac{I_k}{r_{e,c}^t} \right)} - 1 \right) \\ &\quad - \sigma^2. \end{aligned} \tag{35}$$

If the received interference satisfies the inequality (35), the task will be offloaded to the ES through the selected channel to reduce the computational cost; otherwise, the task will be executed locally.

To find the optimal CSD and achieve the NE, we construct a potential function $\varphi(\eta_t)$, which is formulated as

$$\begin{aligned} \phi(\eta_t) = \frac{1}{2} \sum_{u=1}^{U} \sum_{v \in \mathcal{U} \setminus \{u\}} q_u g_u^e q_v g_v^e \mathbb{I}\left(\eta_{u,t} = \eta_{v,t}\right) \mathbb{I}\left(\eta_{u,t} > 0\right) \\ + \sum_{u=1}^{U} q_u g_u^e V_u \mathbb{I}\left(\eta_{u,t} = 0\right) \end{aligned} \tag{36}$$

where

$$V_u = \frac{\frac{q_v g_v^e}{I_k}}{\frac{q_v g_v^e}{2W_u^e \left(\varsigma (f_{l,k})^2 S_k - \left(1 - \beta_k^{(t)}\right) p_k \frac{I_k}{r_{e,c}^t} \right)} - 1} - \sigma^2. \tag{37}$$

This potential function $\varphi(\eta_t)$ can reflect the impact of each user's strategy change on the overall system energy consumption. By minimizing the potential function, we can find the optimal CSD and achieve NE.

Algorithm 1 iteratively and dynamically adjusts the CSD of each MU, ultimately achieving a solution that satisfies all users (i.e., a NE solution). During the iteration process, the algorithm iteratively solves (13), (14), (16), and (17) while computing each user's available TOD set $\Psi_{u,t}$ in conjunction with β_{t+1}^*. The optimal CSD $\eta_{u,t}^*$ is also determined by minimizing $\Gamma_t(\eta_{u,-t}, \eta_{u,t})$. Subsequently, each MU's current CSD $\eta_{u,t}$ is compared with the optimal CSD $\eta_{u,t}^*$. If a more effective CSD is identified, denoted by $\eta_{u,t}^* \neq \eta_{u,t}$, the user will request an update from the ES. The ES randomly grants authorization to one user to update their CSD in response, while those not authorized maintain their current decisions. If no update requests are received, the ES sends an END message. This iterative process continues until an END message is received, which ensures that all users achieve satisfactory CSD.

Algorithm 1 Multiuser CSD Based on β_{t+1}^*

```
Initialization: η_t = 0
for each MU u do
    Measure the interference γ_{k,t}
    Determine the strategy space Ψ_{u,t}
    ▷ Select the optimal CSD
    η*_{u,t} = arg min_{η_{u,t}} Γ_t(η_{u,−t}, η_{u,t})
    ▷ The optimal CSD differs from the current CSD
    if η*_{u,t} ≠ η_{u,t} then
        Send a task to the ES to update its USD
        if an optimal decision is received then
            ▷ Update the MU's CSD to the optimal value
            η_{u,t} = η*_{u,t}
        end if
    end if
    if an END message is received then
        Terminate the process
    end if
end for
return η_t
```

TABLE I
SYSTEM PARAMETERS AND THEIR VALUE SETTINGS

System parameters	Value setting
Number of ESs	[20, 45]
Number of tasks	[40, 80]
Number of MUs	2000
ES's cache size : C	[3, 15] GB
CPU capability of ES	20 GHz
I_{max}	8
D_{max}	8
τ	300 ms
Experience replay memory size: E	1000

V. PERFORMANCE EVALUATION

This section evaluates the performance of the proposed method. It commences with a description of the simulation scenarios and parameter settings, followed by an analysis of the obtained results.

A. Simulation Setup

The considered scenario encompasses a cloud server, an ES, and multiple MU within a 1000 m × 1000 m square area [41]. Consistent with [42] and [43], the channel gain model is represented as $h_(u,t) = w_{u,t} d_u^{-n}$, where d_u^{-n} denotes the distance between MU u and the base station. The input data size for each task, denoted as I_k, ranges uniformly from 1 MB and $I_{\max}$ MB. The service size, denoted as D_k, is randomly chosen between 1 GB and $D_{\max}$ GB. The simulation parameters are chosen based on the MEC network configurations [43], [44]. The main simulation environment settings are summarized in Table I.

We assess the effectiveness of our proposed method by contrasting it with the following benchmark approaches.

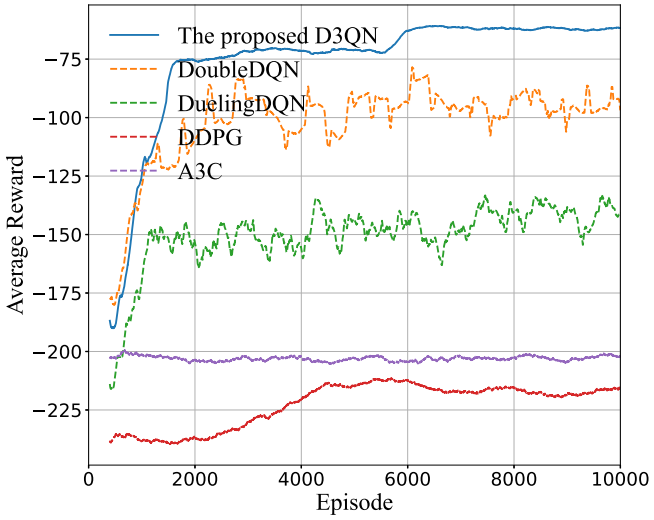

Fig. 4. Convergence comparison of different methods.

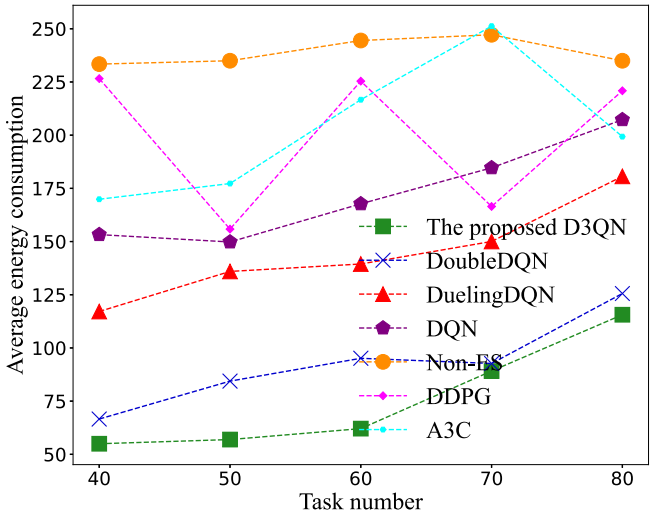

Fig. 5. Impact of task quantity on average energy consumption.

1) *Double DQN [31]:* An offline policy DRL algorithm that begins by collecting the interaction experiences with environments, storing them in an experience pool, and then extracting the sample data for training the Q network.
2) *Dueling DDQN:* To address the potential overestimation issues during DQN training, DDQN introduces two Q networks to enhance the learning stability.
3) *DDPG [30]:* An actor–critic DRL algorithm that utilizes separate networks for policy (actor) and value estimation (critic), enabling the continuous action spaces and improving the decision-making stability.
4) *A3C:* A distributed reinforcement learning algorithm that asynchronously trains the multiple parallel agents to update a shared global model, accelerating convergence and improving robustness in the dynamic environments.

The evaluation network is configured with a learning rate of 10^{-3}, a discount factor of 0.95, and a target network update rate of 0.01. It consists of two hidden layers with 300 and 400 neural neurons, respectively. The mini-batch size is set to 128, and the replay buffer has a capacity of 1000 entries.

B. Experimental Results

Fig. 4 demonstrates the convergence performance of the proposed D3QN algorithm compared to baseline methods, such as DoubleDQN, DuelingDQN, DDPG, and A3C. The proposed D3QN algorithm exhibits a rapid improvement in average reward during the initial 2000 training cycles. It then stabilizes after minor fluctuations, achieving superior convergence performance at a lower level of energy consumption. In contrast, DoubleDQN and DuelingDQN show slower convergence, requiring nearly 3000 cycles to stabilize, with greater fluctuations and lower stability throughout the training process.

The D3QN algorithm amalgamates the strengths of DoubleDQN and DuelingDQN, which enhances the precision of action-value estimations and mitigating estimation errors. By leveraging the improved state and advantage value functions, D3QN achieves faster training velocity and better convergence. This efficiency translates into significant reductions in energy consumption compared to other methods. Furthermore, the incorporation of DDPG and A3C in the evaluation underscores the robustness of D3QN, as it consistently outperforms these advanced algorithms, especially in scenarios with dynamic and complex environments.

Fig. 5 depicts the variations in average energy consumption per time slot for six different algorithms as the number of tasks in the task repository increases. It is evident that the average energy consumption per time slot increases with the number of tasks for all schemes, except for the Non-ES offloading scheme. This increase results from a growing variety of user task requests, which reduce the accuracy of service predictions and the reusability of cached services at the ES. Consequently, more tasks are processed at the CDC, resulting in increased energy consumption. Furthermore, the figure underscores the superior energy consumption management capabilities of the proposed scheme compared to other schemes. For example, when there are 40 tasks, the proposed scheme achieves a 17% energy saving compared to top baseline algorithms (such as DoubleDQN). This substantial reduction in energy consumption stems from the proposed D3QN algorithm's enhanced ability to predict MU task demands accurately. Additionally, with the inclusion of DDPG and A3C in the comparison, it is evident that while these advanced algorithms perform well, they still exhibit higher energy consumption due to their limited adaptability to diverse task distributions and dynamic caching demands. In contrast, the D3QN algorithm consistently demonstrates superior performance by leveraging its precise action-value estimations and efficient caching strategies, achieving lower energy consumption even as task diversity increases.

The results depicted in Fig. 6 demonstrate that the average energy consumption per slot for all schemes decreases

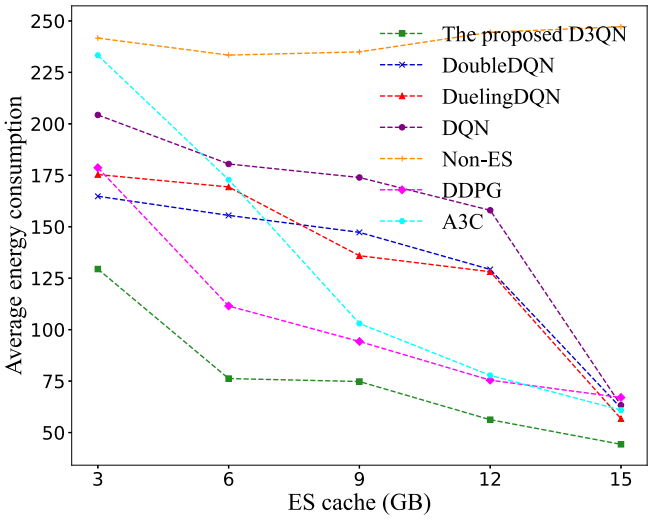

Fig. 6. Impact of ES caching capacity on average energy consumption.

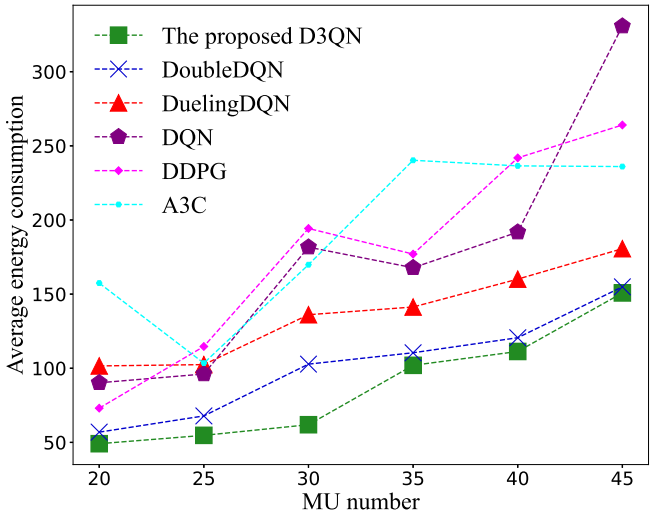

Fig. 7. Impact of mobile user quantity on average energy consumption.

with increasing cache size, except for the Non-ES offloading scheme. This decline can be attributed to the larger cache sizes at the ES, which enable them to store extra services. As a result, the hit rate of tasks requested on ES has improved, which allows more users to perform tasks at a lower cost. When the cache size is sufficiently large to accommodate all task software (exceeding 15 GB), all schemes demonstrate similar performance. In such cases, ES servers are able to cache all task software in the task repository, enabling MUs to carry out tasks via local computing or ES without having to send tasks to the CDC for processing. However, the caching capacity of ES is often constrained in real-world systems, which may prevent the storage of all services. Specifically, our proposed method can save about 60% of energy compared to DoubleDQN when the cache size is 6 GB. Moreover, the DDPG and A3C in the comparison highlights that while these advanced algorithms show better energy management than some baseline methods, they are still outperformed by the proposed D3QN algorithm. DDPG and A3C are less effective at leveraging limited cache capacity to optimize energy efficiency, whereas D3QN dynamically adjusts caching strategies and task distribution, ensuring superior energy savings even under constrained caching scenarios.

Fig. 7 illustrates a rise in average energy consumption across four schemes with an increasing number of MUs. With the cache memory of ES remaining constant, the growing MUs necessitate greater reuse of cached services. The proposed scheme (represented by the green line) consistently demonstrates lower energy consumption throughout the entire range of MUs, highlighting its efficiency in cache resource management and prediction mechanisms. Specifically, at MU = 25, the proposed scheme achieves approximately 19.31% energy savings compared to the best-performing scheme (DoubleDQN). The DQN algorithm exhibits a notable surge in energy consumption when the MU count reaches 45, indicating deficiencies in cache management and task prediction under extreme conditions. With the addition of DDPG and A3C, the comparison further highlights the advantages of the proposed

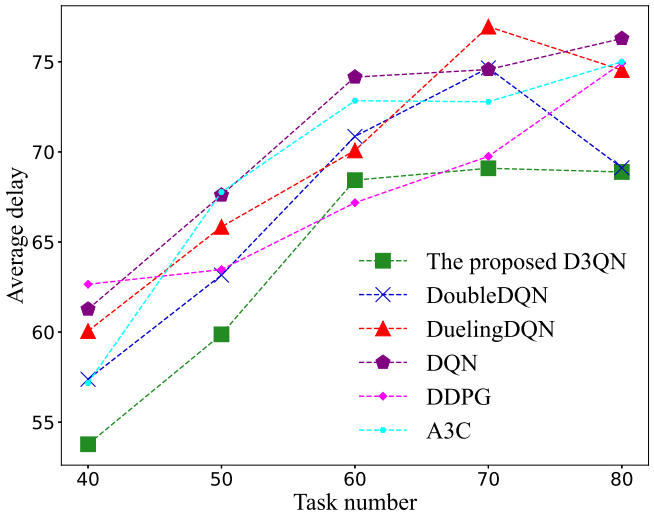

Fig. 8. Impact of task quantity on average delay.

D3QN algorithm. While DDPG and A3C exhibit improved energy management over traditional algorithms like DQN, they are less effective at maintaining energy efficiency as the number of MUs increases. In contrast, D3QN's robust caching and prediction strategies enable it to adapt to the increasing number of MUs while consistently achieving superior energy savings, even under high user density scenarios.

Fig. 8 illustrates that the proposed D3QN algorithm consistently achieves the lowest average delay across various task quantities when compared to DoubleDQN, DuelingDQN, DQN, DDPG, and A3C methods. The enhanced action-value estimation in D3QN enables more accurate service caching and offloading decisions, demonstrating its efficiency in managing higher task loads. While DDPG and A3C show moderate improvements over traditional algorithms, they still exhibit higher delays due to less effective service prediction and task allocation mechanisms. In contrast, D3QN's dynamic decision-making process ensures minimal delay even as task quantities increase.

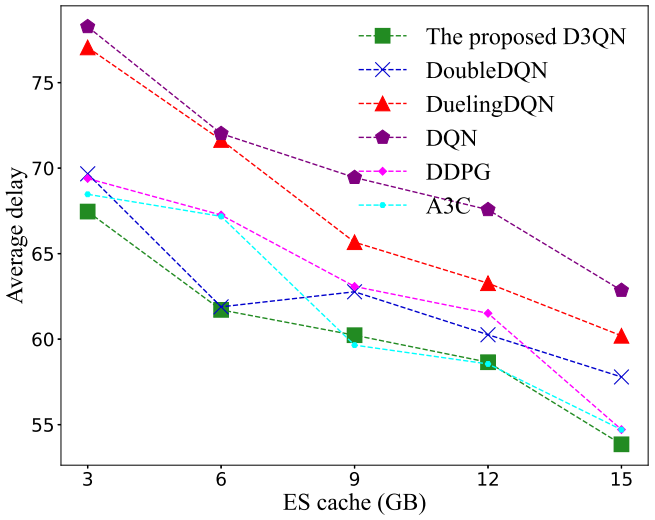

Fig. 9. Impact of ES caching capacity on average delay.

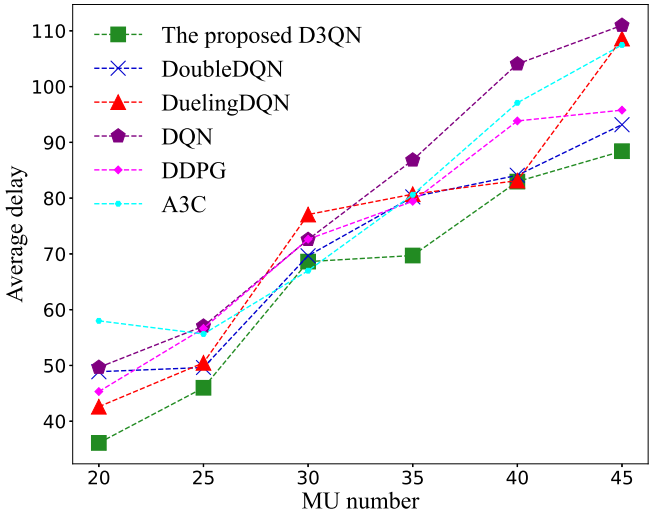

Fig. 10. Impact of mobile user quantity on average delay.

As shown in Fig. 9, the D3QN method significantly outperforms other algorithms in reducing average delay across different ES caching capacities. The results indicate that D3QN effectively leverages available cache resources to optimize task processing, whereas DoubleDQN and DuelingDQN show moderate improvements, DQN exhibits minimal benefits from increased caching capacity, and DDPG and A3C struggle to adapt to constrained caching scenarios. D3QN's superior caching strategies enable it to make efficient use of limited ES resources, minimizing delay even when caching capacity is restricted.

Fig. 10 compares the average delay experienced by different numbers of mobile users. The D3QN algorithm maintains the lowest and most stable delay, showcasing its robustness and scalability. In contrast, DoubleDQN and DuelingDQN face higher delays with fluctuations, and DQN demonstrates substantial delays as the number of users increases, highlighting its limitations in handling larger user bases. DDPG and A3C exhibit improved stability compared to traditional algorithms but are outperformed by D3QN, which dynamically adapts to the increasing number of mobile users, maintaining consistently low delays even under high user densities.

VI. Conclusion

In this article, we have explored the service caching-aided computation offloading issue aimed at enhancing the QoE at edge networks. In order to improve the QoE while considering the energy consumption constraints, we continuously take into account the influential factors, such as caching capacity, computational capabilities, and channel conditions. Due to the complexity of our proposed QoE-based service caching decision process, we have proposed an enhanced D3QN algorithm based on DQN model to identify the optimal service caching mode. By implementing a TLA, our method can enhance the stability of edge networks. Additionally, we also employ a multiuser game model to determine the optimal channel decision-making strategy. The experimental results show that our proposed approach achieves better performance compared to the baseline algorithms.

Future work will focus on two promising directions to further enhance and expand the proposed method. First, we plan to investigate the multiagent reinforcement learning approaches to replace the channel selection optimization with intelligent agents, thereby avoiding the need for complex optimization theory. This direction has the potential to simplify the decision-making process and improve the scalability in multiuser environments. Second, we aim to explore the application of deep streaming reinforcement learning in mobile edge networks. Unlike D3QN, deep streaming reinforcement learning offers a more computationally efficient training mechanism which is better suited for the resource-constrained edge devices, making it particularly valuable for the real-world deployment in edge networks.

Appendix

Computational Complexity Analysis for D3QN

The computational complexity of the D3QN algorithm can be analyzed in two phases: 1) training and 2) inference. Below, we provide a detailed discussion of each phase and compare the complexity of D3QN with other benchmark algorithms, including DQN, DoubleDQN, DDPG, and A3C.

A. Complexity of the Deep Neural Network

The backbone of D3QN algorithm is based on a deep neural network (DNN) which processes the input state and output Q-values for each action. The computational complexity of a DNN is determined by the number of layers, the number of neurons per layer, and the operations required for forward and backward propagation. For a DNN with L layers, where each layer has n neurons, the computational complexity for a single forward pass is approximately $O(n^2L)$. During backpropagation, the gradient computation doubles the complexity, resulting in $O(2n^2L)$. This complexity is shared across all algorithms that rely on DNNs, including DQN, DoubleDQN, DDPG, A3C, and D3QN.

B. Training Phase Complexity

In the training phase, the D3QN employs two DNNs: 1) an evaluation network and 2) a target network, similar to DoubleDQN. However, the dueling architecture adds a state value stream and an advantage stream to the DNN's output layer. This modification does not significantly alter the overall complexity per forward and backward pass, but introduces a small overhead, due to the additional computation of the Q-value as a combination of the state value and advantage values. The complexity per training iteration for D3QN can be expressed as

$$O\left(2n^2L\right)\cdot 2 = O\left(4n^2L\right) \tag{38}$$

where the factor 2 accounts for the evaluation and target networks.

Additionally, the D3QN uses prioritized experience replay, which introduces overhead for maintaining and sampling from the prioritized buffer. The complexity of sampling from the buffer is $O(\log N)$, where N is the buffer size. Updating priorities after each training step also adds a complexity of $O(\log N)$. The complexity per training iteration for D3QN can be expressed as

$$O\left(4n^2L + \log N\right). \tag{39}$$

In comparison:

1) *DQN:* Uses a single network and a uniform replay buffer, with a training complexity of $O(2n^2L)$.
2) *DoubleDQN:* Uses two networks but without the Dueling architecture or prioritized replay, resulting in $O(4n^2L)$ for training.
3) *DDPG:* Employs two networks (actor and critic) without a replay buffer, resulting in a training complexity of $O(4n^2L)$. The actor network determines actions, while the critic network evaluates them, requiring gradient updates for both networks.
4) *A3C:* Uses two networks (actor and critic) and performs parallel updates across P threads, resulting in a training complexity of $O(2Pn^2L)$. Parallelism accelerates training but introduces additional overhead for asynchronous gradient updates.

C. Inference Phase Complexity

In the inference phase, all algorithms use only the evaluation network for action selection. Since only a forward pass is required, the computational complexity is $O(n^2L)$ for all algorithms, including D3QN. This phase is computationally efficient and identical across different approaches, as the architecture of the evaluation network is comparable.

REFERENCES

[1] W. Z. Khan, E. Ahmed, S. Hakak, I. Yaqoob, and A. Ahmed, "Edge computing: A survey," *Future Gener. Comput. Syst.*, vol. 97, pp. 219–235, Aug. 2019.

[2] H. Zhou, Z. Zhang, D. Li, and Z. Su, "Joint optimization of computing offloading and service caching in edge computing-based smart grid," *IEEE Trans. Cloud Comput.*, vol. 11, no. 2, pp. 1122–1132, Apr.–Jun. 2023.

[3] Y. Zhang, X. Lan, J. Ren, and L. Cai, "Efficient computing resource sharing for mobile edge-cloud computing networks," *IEEE/ACM Trans. Netw*, vol. 28, no. 3, pp. 1227–1240, Jun. 2020.

[4] H. Zhou, T. Wu, X. Chen, S. He, and J. Wu, "RAIM: A reverse auctionbased incentive mechanism for mobile data offloading through opportunistic mobile networks," *IEEE Trans. Netw. Sci. Eng.*, vol. 9, no. 6, pp. 3909–3921, Nov. 2022.

[5] Y. Wu, "Cloud-edge orchestration for the Internet of Things: Architecture and AI-powered data processing," *IEEE Internet Things J.*, vol. 8, no. 16, pp. 12792–12805, Aug. 2021.

[6] Z. Ning, X. Kong, F. Xia, W. Hou, and X. Wang, "Green and sustainable cloud of things: Enabling collaborative edge computing," *IEEE Commun. Mag.*, vol. 57, no. 1, pp. 72–78, Jan. 2019.

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

[8] X. Huang, S. Leng, S. Maharjan, and Y. Zhang, "Multi-agent deep reinforcement learning for computation offloading and interference coordination in small cell networks," *IEEE Trans. Veh. Technol.*, vol. 70, no. 9, pp. 9282–9293, Sep. 2021.

[9] H. Wu, Z. Zhang, C. Guan, K. Wolter, and M. Xu, "Collaborate edge and cloud computing with distributed deep learning for smart city Internet of Things," *IEEE Internet Things J.*, vol. 7, no. 9, pp. 8099–8110, Sep. 2020.

[10] K. Jiang, C. Sun, H. Zhou, X. Li, M. Dong, and V. C. M. Leung, "Intelligence-empowered mobile edge computing: Framework, issues, implementation, and outlook," *IEEE Netw.*, vol. 35, no. 5, pp. 74–82, Sep./Oct. 2021.

[11] X. Wang, Y. Han, V. C. M. Leung, D. Niyato, X. Yan, and X. Chen, "Convergence of edge computing and deep learning: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 2, pp. 869–904, 2nd Quart., 2020.

[12] Z. Liao, G. Yin, X. Tang, and P. Liu, "A cooperative community-based framework for service caching and task offloading in multi-access edge computing," *IEEE Trans. Netw. Services Manag.*, vol. 21, no. 3, pp. 3224–3235, Jun. 2024.

[13] T. Mahn, H. Al-Shatri, and A. Klein, "Distributed algorithm for energy efficient joint cloud and edge computing with splittable tasks," in *Proc. IEEE Wireless Commun. Netw. Conf.*, 2019, pp. 1–7.

[14] J. Yan, S. Bi, L. Duan, and Y.-J. A. Zhang, "Pricing-driven service caching and task offloading in mobile edge computing," *IEEE Trans. Wireless Commun.*, vol. 20, no. 7, pp. 4495–4512, Jul. 2021.

[15] Z. Cui, X. Shi, Z. Zhang, W. Zhang, and J. Chen, "Many-objective joint optimization of computation offloading and service caching in mobile edge computing," *Simul. Model. Pract. Theory*, vol. 133, May 2024, Art. no. 102917, doi: 10.1016/j.simpat.2024.102917.

[16] C. Cheng, L. Zhai, X. Zhu, Y. Jia, and Y. Li, "Dynamic task offloading and service caching based on game theory in vehicular edge computing networks," *Comput. Commun.*, vol. 224, pp. 29–41, Aug. 2024, doi: 10.1016/j.comcom.2024.05.020.

[17] Z. Ning et al., "Intelligent edge computing in internet of vehicles: A joint computation offloading and caching solution," *IEEE Trans. Intell. Transp.Syst.*, vol. 22, no. 4, pp. 2212–2225, Apr. 2021.

[18] Z. Chen, Z. Chen, and Y. Jia, "Integrated task caching, computation offloading and resource allocation for mobile edge computing," in *Proc. IEEE GLOBECOM*, 2019, pp. 1–6.

[19] Q. Zhang, W. Xu, H. Luo, and S. Luo, "Fast globally optimal computational offloading and service caching in container-based edge computing systems," *IEEE Internet Things J.*, vol. 11, no. 13, pp. 23780–23792, Jul. 2024.

[20] Y. Yang, Y. Hu, and M. C. Gursoy, "Deep reinforcement learning and optimization based green mobile edge computing," in *Proc. IEEE CCNC*, Las Vegas, NV, USA, Jan. 2021, pp. 1–2.

[21] X. Dai, S. Tian, H. Liu, Z. Li, H. Jiang, and Q. Deng, "Joint optimization of offloading and caching in full-duplex-enabled edge computing networks," *IEEE Trans. Mobile Comput.*, early access, Feb. 27, 2025, doi: 10.1109/TMC.2025.3546263.

[22] M. Zhao, R. Zhang, Z. He, and K. Li, "Joint optimization of trajectory, offloading, caching, and migration for UAV-assisted MEC," *IEEE Trans. Mobile Comput.*, vol. 24, no. 3, pp. 1981–1998, Mar. 2025, doi: 10.1109/TMC.2024.3486995.

[23] H. Tang, H. Wu, G. Qu, and R. Li, "Double deep Q-network based dynamic framing offloading in vehicular edge computing," *IEEE Trans. Netw. Sci. Eng.*, vol. 10, no. 3, pp. 1297–1310, May/Jun. 2023.
[24] M. Wu, W. Chen, L. Qian, L. Guo, and I. Lee, "Joint service caching and secure computation offloading for reconfigurable-intelligent-surface-assisted edge computing networks," *IEEE Internet Things J.*, vol. 11, no. 19, pp. 30469–30482, Oct. 2024, doi: 10.1109/JIOT.2024.3404972.
[25] H. Liu and G. Cao, "Deep reinforcement learning-based server selection for mobile edge computing," *IEEE Trans. Veh. Technol.*, vol. 70, no. 12, pp. 13351–13363, Dec. 2021, doi: 10.1109/TVT.2021.3124127.
[26] Z. Zhang, N. Wang, H. Wu, C. Tang, and R. Li, "MR-DRO: A fast and efficient task offloading algorithm in heterogeneous edge/cloud computing environments," *IEEE Internet Things J.*, vol. 10, no. 4, pp. 3165–3178, Feb. 2023.
[27] S. Li, B. Li, and W. Zhao, "Joint optimization of caching and computation in multi-server NOMA-MEC system via reinforcement learning," *IEEE Access*, vol. 8, pp. 112762–112771, 2020.
[28] M. Xie, J. Ye, G. Zhang, and X. Ni, "Deep reinforcement learning-based computation offloading and distributed edge service caching for mobile edge computing," *Comput. Netw.*, vol. 250, Aug. 2024, Art. no. 110564, doi: 10.1016/j.comnet.2024.110564.
[29] N. Lin, X. Han, A. Hawbani, Y. Sun, Y. Guan, and L. Zhao, "Deep reinforcement learning based dual-timescale service caching and computation offloading for multi-UAV assisted MEC systems," *IEEE Trans. Netw. Services Manag.*, vol. 22, no. 1, pp. 605–617, Feb. 2025, doi: 10.1109/TNSM.2024.3468312.
[30] L. Zhao, Z. Zhao, A. Hawbani, Z. Liu, Z. Tan, and K. Yu, "Dynamic caching dependency-aware task offloading in mobile edge computing," *IEEE Trans. Comput.*, early access, Jan. 23, 2025, doi: 10.1109/TC.2025.3533091.
[31] C. Sun, X. Li, C. Wang, Q. He, X. Wang, and V. C. M. Leung, "Hierarchical deep reinforcement learning for joint service caching and computation offloading in mobile edge-cloud computing," *IEEE Trans. Services Comput.*, vol. 17, no. 4, pp. 1548–1564, Jul./Aug. 2024.
[32] Y. Xu, Z. Peng, N. Song, Y. Qiu, C. Zhang, and Y. Zhang, "Joint optimization of service caching and task offloading for customer application in MEC: A hybrid SAC scheme," *IEEE Trans. Consum. Electron.*, early access, Aug. 13, 2024, doi: 10.1109/TCE.2024.3443168.
[33] Q. Fan, W. Zhang, C. Ling, R. Yadav, D. Wang, and H. He, "Mobility-aware cooperative service caching for mobile augmented reality services in mobile edge computing," *IEEE Trans. Veh. Technol.*, vol. 73, no. 11, pp. 17543–17557, Nov. 2024. doi: 10.1109/TVT.2024.3422179.
[34] R. Zheng, H. Wang, M. De Mari, M. Cui, X. Chu, and T. Q. S. Quek, "Dynamic computation offloading in ultra-dense networks based on mean field games," *IEEE Trans. Wireless Commun.*, vol. 20, no. 10, pp. 6551–6565, Oct. 2021.
[35] A. P. Miettinen and J. K. Nurminen, "Energy efficiency of mobile clients in cloud computing," in *Proc. HotCloud*, 2010, pp. 1–7.
[36] W. Wen, Y. Cui, T. Q. Quek, F.-C. Zheng, and S. Jin, "Joint optimal software caching, computation offloading and communications resource allocation for mobile edge computing," *IEEE Trans. Veh. Technol.*, vol. 69, no. 7, pp. 7879–7894, Jul. 2020.
[37] M. Wu, K. Li, L. Qian, Y. Wu, and I. Lee, "Secure computation offloading and service caching in mobile edge computing networks," *IEEE Commun. Lett.*, vol. 28, no. 2, pp. 432–436, Feb. 2024.
[38] R. Zhou et al., "User preference oriented service caching and task offloading for UAV-assisted MEC networks," *IEEE Trans. Serv. Comput.*, early access, Feb. 13, 2025, doi: 10.1109/TSC.2025.3536319.
[39] H. Jiang, J. Cai, Z. Xiao, K. Yang, H. Chen, and J. Liu, "Vehicle-assisted service caching for task offloading in vehicular edge computing," *IEEE Trans. Mobile Comput.*, early access, Feb. 25, 2025, doi: 10.1109/TMC.2025.3545444.
[40] T. S. Rappaport, *Wireless Communications: Principles and Practice*, vol. 2. New Jersey, NJ, USA: Prentice Hall, 1996.
[41] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Trans. Netw.*, vol. 24, no. 5, pp. 2795–2808, Oct. 2016.
[42] Y. Qian, R. Wang, J. Wu, B. Tan, and H. Ren, "Reinforcement learning-based optimal computing and caching in mobile edge network," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 10, pp. 2343–2355, Oct. 2020.
[43] Y. Sun, Y. Cui, and H. Liu, "Joint pushing and caching for bandwidth utilization maximization in wireless networks," *IEEE Trans. Commun.*, vol. 67, no. 1, pp. 391–404, Jan. 2019.
[44] Z. Chen, W. Yi, A. S. Alam, and A. Nallanathan, "Dynamic task software caching-assisted computation offloading for multi-access edge computing," *IEEE Trans. Commun.*, vol. 70, no. 10, pp. 6950–6965, Oct. 2022.

Bo Xie received the M.S. degree in computer technology from Guizhou University, Guiyang, China, in 2022. He is currently pursuing the Ph.D. degree in electronic science and technology with South China Normal University, Guangzhou, China.

His current research interests include mobile-edge computing and optimization, edge intelligence.

Jinhua Xie is currently pursuing the M.S. degree with the School of Electronic and Information Engineering, South China Normal University, Guangzhou, China, in 2025.

His current research interests are in the areas of mobile-edge computing and optimization, edge intelligence.

Haixia Cui (Senior Member, IEEE) received the M.S. and Ph.D. degrees in communication engineering from South China University of Technology, Guangzhou, China, in 2005 and 2011, respectively.

She is currently a Full Professor with the School of Electronic Science and Engineering, South China Normal University, Guangzhou, China. From July 2014 to July 2015, she was an Advanced Visiting Scholar and Visiting Associate Professor with the Department of Electrical and Computer Engineering, the University of British Columbia, Vancouver, BC, Canada. She has authored or co-authored more than 70 refereed journal and conference papers and one books. She also holds about 30 patents. Her current research interests are in the areas of mobile-edge computing, vehicular networks, cooperative communication, wireless resource allocation, 5G/6G, multiple access control, and power control in wireless networks.

Yejun He (Senior Member, IEEE) received the Ph.D. degree in information and communication engineering from the Huazhong University of Science and Technology, Wuhan, China, in 2005.

From 2005 to 2006, he was a Research Associate with the Department of Electronic and Information Engineering, The Hong Kong Polytechnic University, Hong Kong. From 2006 to 2007, he was a Research Associate with the Department of Electronic Engineering, Faculty of Engineering, The Chinese University of Hong Kong, Hong Kong. In 2012, he joined the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON, Canada, as a Visiting Professor. From 2013 to 2015, he was an Advanced Visiting Scholar and a Visiting Professor with the School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, USA. From 2023 to 2024, he is an Advanced Research Scholar and a Visiting Professor with the Department of Electrical and Computer Engineering, National University of Singapore, Singapore. Since 2006, he has been a Faculty of Shenzhen University, Shenzhen, China, where he is currently a Full Professor with the College of Electronics and Information Engineering, Shenzhen University, the Director of Sino–British Antennas and Propagation Joint Laboratory of Ministry of Science and Technology of the People's Republic of China, the Guangdong Engineering Research Center of Base Station Antennas and Propagation, and the Shenzhen Key Laboratory of Antennas and Propagation. He was selected as a Leading Talent in the "Guangdong Special Support Program" and the Shenzhen "Pengcheng Scholar" Distinguished Professor, China, in 2024 and 2020, respectively. He has authored or co-authored more than 300 refereed journal and conference papers and seven books. He holds about 20 patents. His research interests include wireless communications, antennas, and radio frequency.

Dr. He was also a recipient of the Shenzhen Overseas High-Caliber Personnel Level B (Peacock Plan Award B) and Shenzhen High-Level Professional Talent (Local Leading Talent). He received the Second Prize of Shenzhen Science and Technology Progress Award in 2017, the Three Prize of Guangdong Provincial Science and Technology Progress Award in 2018, the Second Prize of Guangdong Provincial Science and Technology Progress Award in 2023, and the 10th Guangdong Provincial Patent Excellence Award in 2023. He is currently the Chair of IEEE Antennas and Propagation Society-Shenzhen Chapter and obtained the 2022 IEEE APS Outstanding Chapter Award. He has served as a Technical Program Committee Member or a Session Chair for various conferences, including the IEEE Global Telecommunications Conference, the IEEE International Conference on Communications, the IEEE Wireless Communication Networking Conference, and the IEEE Vehicular Technology Conference. He served as the TPC Chair for IEEE ComComAp 2021 and the General Chair for IEEE ComComAp 2019. He was selected as a Board Member of the IEEE Wireless and Optical Communications Conference (WOCC). He served as the TPC Co-Chair for WOCC 2023/2022/2019/2015, APCAP 2023, UCMMT 2023, ACES-China2023, and NEMO 2020. He acted as the Publicity Chair of several international conferences, such as the IEEE PIMRC 2012. He is serving as an Executive Chair of 2024 IEEE International Workshop of Radio Frequency and Antenna Technologies. He is the Principal Investigator for over 40 current or finished research projects, including the National Natural Science Foundation of China, the Science and Technology Program of Guangdong Province, and the Science and Technology Program of Shenzhen City. He has served as a Reviewer for various journals, such as the IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, IEEE TRANSACTIONS ON COMMUNICATIONS, IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS, IEEE TRANSACTIONS ON ANTENNAS AND PROPAGATION, IEEE WIRELESS COMMUNICATIONS, IEEE COMMUNICATIONS LETTERS, *the International Journal of Communication Systems*, and *Wireless Personal Communications*. He is serving as an Associate Editor for IEEE TRANSACTIONS ON ANTENNAS AND PROPAGATION, IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, IEEE TRANSACTIONS ON MOBILE COMPUTING, *IEEE Antennas and Propagation Magazine*, IEEE ANTENNAS AND WIRELESS PROPAGATION LETTERS, *International Journal of Communication Systems*, *China Communications*, and *ZTE Communications*. He is a Fellow of IET, a Senior Member of the China Institute of Communications, and a Senior Member of the China Institute of Electronics.

Mohsen Guizani (Fellow, IEEE) received the B.S. (with Distinction), M.S., and Ph.D. degrees in electrical and computer engineering from Syracuse University, Syracuse, NY, USA, in 1985, 1987, and 1990, respectively.

He is currently a Professor of Machine Learning with Mohamed Bin Zayed University of Artificial Intelligence, Abu Dhabi, UAE. Previously, he worked in different institutions in the USA. He was listed as a Clarivate Analytics Highly Cited Researcher in Computer Science in 2019, 2020, 2021, and 2022. He is the author of 11 books, more than 1000 publications and several U.S. patents. His research interests include applied machine learning and artificial intelligence, smart city, Internet of Things (IoT), intelligent autonomous systems, and cybersecurity.

Dr. Guizani has won several research awards, including the "2015 IEEE Communications Society Best Survey Paper Award," the Best ComSoc Journal Paper Award in 2021 as well five Best Paper Awards from ICC and Globecom Conferences. He is also the recipient of the 2017 IEEE Communications Society Wireless Technical Committee Recognition Award, the 2018 AdHoc Technical Committee Recognition Award, and the 2019 IEEE Communications and Information Security Technical Recognition Award. He served as the Editor-in-Chief for IEEE Network and is currently serving on the Editorial Boards of many IEEE Transactions and Magazines. He was the Chair of the IEEE Communications Society Wireless Technical Committee and the Chair of the TAOS Technical Committee. He served as the IEEE Computer Society Distinguished Speaker and is currently the IEEE ComSoc Distinguished Lecturer.