Omni-DDPG-Based Secure Computation Offloading in Collaborative Mobile Edge Computing

Xue Qin, *Student Member, IEEE*, Xinyu Huang, *Member, IEEE*, Shisheng Hu, *Member, IEEE*, and Xuemin Shen, *Fellow, IEEE*

***Abstract*—In this paper, we investigate secure and efficient computation offloading in mobile edge computing (MEC) systems. Particularly, computation tasks are dynamically partitioned into multiple sub-tasks for parallel processing on both local devices and edge servers to reduce service latency. In addition, the friendly jamming technique is applied to degrade the interception capabilities of eavesdroppers to protect data secrecy. Our objective is to maximize the number of tasks completed before their respective deadlines and, at the same time, to minimize energy consumption and service latency with security guarantee. To simultaneously handle heterogeneous offloading decisions, we propose an omni-deep deterministic policy gradient (Omni-DDPG) approach that integrates a variational autoencoder for discrete jammer selection, an Ornstein-Uhlenbeck process for continuous computing power allocation, and a Dirichlet distribution for constrained continuous task partitioning. The proposed approach has a low complexity growing linearly with the system size, maintains light memory usage, and enables real-time decisions without using complicated optimization solvers. Extensive simulation results demonstrate our proposed approach can achieve better performance in terms of the number of completed tasks before expiration, energy consumption, and service latency, while satisfying secrecy requirements, compared with the benchmarks such as DDPG, deep Q-network, and greedy algorithms.**

***Index Terms*—Mobile edge computing, computation offloading, task partitioning, physical layer security, deep deterministic policy gradients.**

I. Introduction

MODERN Internet of Things (IoT) systems are rapidly evolving toward higher automation and intelligence to enable more advanced applications in manufacturing, healthcare, and transportation [1]. The evolution is accompanied by increasingly complex system functionalities, including environment perception and system control, which demand substantial real-time and computation-intensive data processing. However, most IoT devices remain resource-constrained, posing a significant challenge to fulfilling the demand. To address this challenge, mobile edge computing (MEC) servers deployed at the edge of the wireless networks, e.g., base stations, can significantly improve computational performance with low latency [2], [3]. By offloading tasks from IoT devices to edge servers in proximity, computation offloading reduces service latency and mitigates network congestion [4].

To reduce task processing latency and improve resource utilization efficiency in MEC systems, the collaborative MEC paradigm has recently received increasing attention. In this paradigm, multiple edge servers distributed across neighboring geographic regions collaboratively provide the edge computing service to individual IoT devices [5]. Particularly, computation tasks, such as video analytics and speech recognition, can be partitioned into multiple subtasks, which are offloaded to multiple servers for parallel processing. In this way, the task processing latency can be reduced, and the system resilience in response to server failures can be improved. In addition, by incorporating partial offloading, some of the subtasks can be allocated to IoT devices for processing such that the on-device computing resources can be efficiently utilized to further enhance task processing performance in MEC systems [6].

Despite the potential of collaborative MEC paradigm, there are two challenges in computation offloading for satisfactory MEC service provision. First, the computing service provision in modern IoT systems is typically subject to stringent service requirements, including low latency, high reliability, and high energy efficiency, while MEC network conditions, such as computation demand of IoT devices and resource availability of servers, can be highly dynamic [7]. Second, due to the broadcast nature of wireless communications, during computation offloading, the data in computation tasks are highly vulnerable to being wiretapped by nearby eavesdroppers [8]. Without proper protection mechanisms, private data (e.g., health and identity information) may be intercepted [9], discouraging privacy-sensitive IoT users from utilizing MEC.

To address the challenge of network dynamics, optimization and data-driven methods for computation offloading have been proposed in the literature. Some conventional optimization methods such as mixed integer non-linear programming (MINLP) [10], successive convex approximation (SCA) [11] and Lyapunov optimization [12], have been adopted to minimize latency and energy consumption. However, the optimization is based on a fixed network snapshot, and any change in network dynamics necessitates re-deriving and re-solving the problem, which limits its applicability to high-dimensional observations in task partitioning and offloading. Data-driven machine learning [13] and deep learning (DL) have been used to learn

Received 26 August 2025; revised 22 December 2025; accepted 17 February 2026. Date of publication 23 February 2026; date of current version 6 July 2026. Recommended for acceptance by X. Chen. *(Corresponding authors: Xinyu Huang; Shisheng Hu.)*

The authors are with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON N2L 3G1, Canada (e-mail: x7qin@uwaterloo.ca; xinyu.huang1@uwaterloo.ca; s97hu@uwaterloo.ca; sshen@uwaterloo.ca).

Digital Object Identifier 10.1109/TMC.2026.3667453

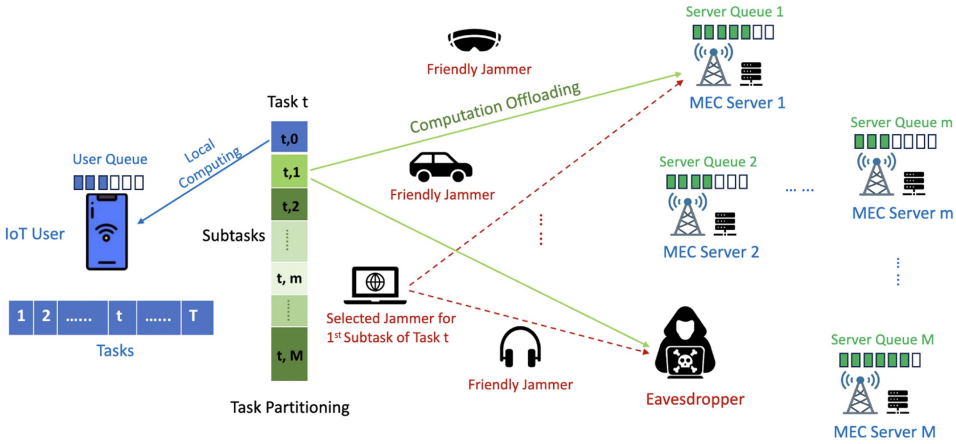

Fig. 1. System model.

from historical labeled data, predict and optimize computational offloading decisions [14]. However, it is difficult to obtain such labeled data, due to the complex and dynamic task partitioning and optimization environment in MEC networks.

For addressing the aforementioned issues, deep reinforcement learning (DRL) technique is recently exploited to learn optimal computation offloading policies through interaction with environments without requiring an explicit system model, making it well suited in dynamic MEC networks [15], [16]. To handle discrete action spaces (e.g., deciding whether tasks are offloaded and selecting destination edge servers), existing DRL algorithms with value-based functions are typically based on the deep Q-network (DQN) [17]. While, for continuous action spaces (e.g., partial offloading ratio determination and computing power control), some algorithms are based on the policy gradient in actor-critic structures, such as the deep deterministic policy gradient (DDPG) [18], [19] and Soft Actor Critic (SAC) [20]. For partial offloading and resource allocation, the DRL and optimization techniques could be used to deal with different decision variables separately [21]. Moreover, the hybrid discrete-continuous actions could be handled with relaxation of the continuous action space in an end-to-end manner to improve overall system performance [22]. During computation task partitioning, the sum of the percentage of all subtasks of a given task should be equal to one. To satisfy the constrained proportional action spaces, the softmax function and Dirichlet distribution have been incorporated into DDPG algorithm to optimize resource allocation and improve service quality [23], [24].

Physical layer security (PLS) [25] offers a lightweight and information-theoretic way. Without the requirements of encryption keys and intensive computation, PLS has been considered as a complementary promising technology to traditional cryptographic methods [26] to resolve the eavesdropping problem for resource-constrained IoT devices. As a PLS technique, friendly jamming involves trusted nodes deliberately transmitting artificial noise to interfere with potential eavesdroppers [27]. This degrades the interception capabilities of eavesdroppers while causing minimal impact on legitimate receivers. By applying friendly jamming, computation tasks can be securely offloaded to the target MEC servers at a secrecy transmission rate, thereby preventing eavesdroppers from successfully recovering the information [28]. However, the integration of friendly jamming into task partitioning and offloading introduces a more complex discrete-continuous action space, which may not be effectively handled by the DRL methods for computation offloading in the literature.

In this paper, we investigate the secure and efficient offloading in dynamic MEC systems in the presence of eavesdroppers. Partial offloading in multi-server collaborative MEC systems is considered, allowing computation tasks to be partitioned into multiple subtasks for simultaneously computing on local devices and edge servers. During computation offloading, to protect the transmitted data, friendly jammers are dynamically selected from candidate nodes to broadcast artificial noise to interfere with the eavesdropper, degrading the reception quality of the eavesdropper. We jointly optimize task partitioning, jammer selection, and computational resource allocation to achieve multiple objectives, including energy efficiency maximization, latency minimization, and security guarantee. This problem is greatly challenging, as we need to deal with different types of actions simultaneously, including constrained and continuous action space (partial offloading ratio and computation resource allocation), and discrete action space (jammer selection). The existing DRL methods struggle with hybrid action spaces in secure MEC task offloading, as they handle either discrete or continuous actions and fail to capture the interdependence and constraints among resource allocation variables, such as partition ratios summing to one. To this end, we propose an improved DDPG approach to overcome these limitations by integrating the Dirichlet distribution, variational autoencoders (VAE), and Ornstein-Uhlenbeck (OU) process to handle all possible types of action spaces for task partition and resource allocation in MEC systems, named Omni-DDPG (ODDPG). The proposed ODDPG approach achieves real-time decision-making with only linear growth in computational complexity, low memory consumption, and no dependence on complicated solver calls, while consistently outperforming existing DRL methods. Moreover, this approach can enable intelligent and secure IoT-MEC applications, such as industrial IoT monitoring, autonomous

vehicular networks, and privacy-sensitive healthcare systems, by supporting latency-sensitive and energy-constrained task offloading while ensuring data confidentiality.

In a nutshell, the main contributions of this paper can be summarized as follows:

- A PLS-assisted partial offloading model in multi-server collaborative MEC system is devised to ensure offloading security and improve offloading efficiency in the presence of eavesdroppers.
- A novel end-to-end DRL approach, ODDPG, is developed by integrating VAE and Dirichlet distribution into DDPG to dynamically handle all types of action space, including continuous task partitioning ratio determination with constraints, discrete jammer selection, and continuous CPU/GPU frequency allocation at edge server. The complexity of ODDPG grows linearly with the number of edge servers and jammer candidates, and its solver-free design ensures the scalability to large-scale MEC systems.
- Extensive simulation results are provided, which demonstrate that the proposed ODDPG approach outperforms benchmarks such as DDPG, DQN and greedy algorithm, in terms of the number of completed tasks before expiration, energy consumption, and service latency, while satisfying secrecy requirements.

The remainder of the paper is organized as follows. The system model and problem formulation are presented in Section II. Then, the ODDPG approach is proposed in Section III, and the simulation results are provided in Section IV. Finally, we conclude this work in Section V.

II. System Model and Problem Formulation

A. System Model

As shown in Fig. 1, we consider a scenario, where an IoT device generates and partitions computing tasks into multiple subtasks for executing in parallel using local computing resources and on multiple edge servers. There is one nearby eavesdropper that can intercept offloaded tasks.[1] In addition, the eavesdropper is assumed to be stationary.[2] To address the eavesdropping issue, PLS is adopted to secure latency-sensitive wireless task offloading by leveraging the achievable secrecy transmission rate, avoiding the computational and energy overhead of conventional encryption on resource-constrained IoT devices. There are friendly jammers that can be selected to transmit artificial noise to interfere with the eavesdropper and protect the information from being decoded by the eavesdropper. Particularly, a single-jammer protection strategy is considered, where the offloading of each subtask is assisted by one friendly jammer.[3]

[1]Extending the system model to scenarios with multiple eavesdroppers would introduce additional coupled secrecy constraints, requiring joint consideration of worst-case or aggregate information leakage across interception links.

[2]Extending the system model to scenarios with mobile eavesdroppers requires addressing the additional challenge posed by time-varying interception channels, which is left for future work.

[3]Activating multiple jammers simultaneously for a single subtask could potentially improve secrecy performance but substantially increase energy consumption and coordination complexity. In this paper, the single-jammer strategy is adopted to balance the secrecy enhancement with energy efficiency and coordination simplicity.

At time slot t, the status of the IoT user and MEC servers are denoted as $u_t = (Q_0, f_0^{\max})$ and $S_t = \{s_1, \ldots, s_m, \ldots, s_M\}$, where $s_m = (Q_m, f_m^{\max})$. Q_0 represents the number of tasks in task queue of the IoT user, and Q_m represents that of the m^{th} edge server, where $m \in \mathcal{M} = \{1, 2, \ldots, M\}$. The $f_0^{\max}$ and $f_m^{\max}$ are their maximum CPU frequencies, respectively. The task Γ_t is denoted by $\Gamma_t = \{d_t, c_t, T_t^{\max}\}$, where d_t, c_t and $T_t^{\max}$ represent the data size, requested CPU cycles and deadline of the task, respectively. When the task Γ_t needs to be processed, it can be partitioned into subtasks and each subtask corresponds to a specific portion with partition ratio $p_{t,0}, p_{t,m} \in [0, 1]$ and $p_{t,0} + \sum_{m=1}^{M} p_{t,m} = 1$, where $p_{t,0}$ and $p_{t,m}$ are the partition ratios of subtasks processed by the user and m^{th} edge server, respectively. It is possible that there is no subtask offloaded to the m^{th} edge server. During computation offloading, the m^{th} subtask of the task Γ_t is offloaded to the m^{th} edge server with the aid of a selected friendly jammer $J_{t,m}$, which interferes eavesdropper with artificial noise for security purposes. Then, a central controller determines the CPU frequency at the designated edge server to execute the m^{th} subtask of the t^{th} task, represented by $f_{t,m}$.

B. Time Cost for Completing Task Γ_t

The time cost of all subtasks of task Γ_t can be denoted as a vector $T_t = (T_{t,0}, \ldots, T_{t,m}, \ldots, T_{t,M})$, where $T_{t,0}$ is the time cost to complete the subtask at IoT user, and $T_{t,m}$ is that to complete the m^{th} subtask of the task Γ_t at the m^{th} edge server. The $T_{t,0}$ and $T_{t,m}$ are given by

$$T_{t,0} = T_{t,0}^{\text{wait}} + T_{t,0}^{\text{comp}}, \tag{1}$$

$$T_{t,m} = T_{t,m}^{\text{tran}} + T_{t,m}^{\text{wait}} + T_{t,m}^{\text{comp}}, \tag{2}$$

where, $T_{t,m}^{\text{tran}}$ is secure transmission time; $T_{t,0}^{\text{wait}}$ and $T_{t,m}^{\text{wait}}$ are waiting time; and $T_{t,0}^{\text{comp}}$ and $T_{t,m}^{\text{comp}}$ are computing time.

The secure transmission time $T_{t,m}^{\text{tran}}$ can be obtained by dividing the subtask data size $p_{t,m}d_t$ by the secrecy transmission rate $r_{t,m}^{\text{sec}}$

$$T_{t,m}^{\text{tran}} = \frac{p_{t,m}d_t}{r_{t,m}^{\text{sec}}}, \tag{3}$$

where the secrecy transmission rate is the rate at which the information is transmitted to the desired destination, while eavesdroppers cannot decode any useful information. As the secrecy transmission rate is inversely proportional to the secure transmission time, it directly influences the optimal offloading decision and service latency. A higher secrecy transmission rate enables more subtasks to be safely and efficiently offloaded to MEC servers with lower service latencies. Conversely, due to adverse device-server channel conditions or strong eavesdropper channels, a lower secrecy transmission rate may result in higher delay, more local computation or the adjustment of jammer selection. Based on previous studies [25], [29], the secrecy transmission rate $r_{t,m}^{\text{sec}}$ can be calculated as follows:

$$r_{t,m}^{\text{sec}} = r_{t,m}^{s} - r_{t,m}^{e}, \tag{4}$$

where $r^s_{t,m}$ is the transmission rate at the m^{th} edge server and $r^e_{t,m}$ is the transmission rate at the eavesdropper.

The transmission rate $r^s_{t,m}$ can be calculated as

$$r^s_{t,m} = B_{t,m} log_2 \left(1 + \frac{P_{u_{t,m}} h_{u,s_m} L_{u,s_m}}{N_0 + P_{J_{t,m}} h_{J_{t,m},s_m} L_{J_{t,m},s_m}}\right), \quad (5)$$

where, $B_{t,m}$ is the bandwidth; $P_{u_{t,m}}$ and $P_{J_{t,m}}$ are the transmission power of the IoT user and the selected friendly jammer, respectively; the h_{u,s_m} and L_{u,s_m} are small-scale fading and path loss for the channel between the IoT user and m^{th} edge server; and $h_{J_{t,m},s_m}$ and $L_{J_{t,m},s_m}$ are those for the channel between the selected jammer and the m^{th} edge server.

Similarly, the transmission rate $r^e_{t,m}$ is given by

$$r^e_{t,m} = B_{t,m} log_2 \left(1 + \frac{P_{u_{t,m}} h_{u,e} L_{u,e}}{N_0 + P_{J_{t,m}} h_{J_{t,m},e} L_{J_{t,m},e}}\right). \quad (6)$$

To calculate the data rate in (5), the device-server channel state information (CSI), i.e., $h_{u,s_m} L_{u,s_m}$, is estimated via standard uplink pilot transmission. However, the stationary eavesdropper does not engage in the channel estimation. Therefore, we assume that only the statistical CSI of the device-eavesdropper channel is known. Note that, there may be correlation between the device-server and device-eavesdropper channels when the eavesdropper is geographically close to the MEC server [30]. For analytical tractability, we assume the two channels experience independent small-scale fadings in calculating the corresponding data rates (5)–(6).

Before a subtask can be processed, it must wait until all previously queued tasks have been completed. The waiting time $T^{\text{wait}}_{t,m}$ includes the residual running time of the subtask which is currently being processed $T^{\text{res}}_{t,m}$ and the queuing time $T^{\text{que}}_{t,m}$:

$$T^{\text{wait}}_{t,m} = T^{\text{res}}_{t,m} + T^{\text{que}}_{t,m}, \quad (7)$$

where, $T^{\text{res}}_{t,m}$ is the difference between the estimated completion time $T^c_{t,m}$ and the start time $T^s_{t,m}$ of the in-process subtask:

$$T^{\text{res}}_{t,m} = T^c_{t,m} - T^s_{t,m}. \quad (8)$$

We assume there are $N_{t,m}$ subtasks in the queue Q_m when the m^{th} subtask of the task Γ_t arrives. The computing time for a particular subtask can be obtained by dividing the required CPU cycles $p_{*,j} c_*$ by the allocated CPU frequency $f_{*,m}$. Therefore, the queuing time $T^{\text{que}}_{t,m}$ of the m^{th} subtask of the task Γ_t is the aggregated computing time of all the subtasks before it in the queue, namely

$$T^{\text{que}}_{t,m} = \sum_{1}^{N_{t,m}} \frac{p_{*,m} c_*}{f_{*,m}}. \quad (9)$$

The required computing time of the m^{th} subtask of the task Γ_t is calculated as

$$T^{\text{comp}}_{t,m} = \frac{p_{t,m} c_t}{f_{t,m}}, \quad (10)$$

where, $f_{t,m}$ denotes the allocated CPU frequency of m^{th} edge server for the m^{th} subtasks of task Γ_t.

Similarly, $T^{\text{res}}_{t,0}$, $T^{\text{que}}_{t,0}$ and $T^{\text{comp}}_{t,0}$ can be calculated as

$$T^{\text{res}}_{t,0} = T^c_{t,0} - T^s_{t,0}, \quad (11)$$

$$T^{\text{que}}_{t,0} = \sum_{1}^{N_{t,0}} \frac{p_{*,0} c_*}{f_{*,0}}, \quad (12)$$

$$T^{\text{comp}}_{t,0} = \frac{p_{t,0} c_t}{f_{t,0}}. \quad (13)$$

For task Γ_t, the total completion time T_t is the longest time spent among each subtask process time, is given by

$$T_t = \max\left(T_{t,0}, \ldots, T_{t,m}, \ldots, T_{t,M}\right). \quad (14)$$

The task Γ_t is successfully completed only when its completion time T_t is equal or less than its deadline $T^{\max}_t$. The indicator of the success of task Γ_t is denoted by ν_t and calculated as

$$\nu_t = \begin{cases} +1, & T_t \leq \text{T}^{\max}_t \\ 0, & \text{otherwise.} \end{cases} \quad (15)$$

C. Energy Consumption for Completing Task Γ_t

For the subtask which is processed by the user, the secure transmission energy consumption $E^{\text{tran}}_{t,0}$ can be considered as 0. For the m^{th} subtask of the task Γ_t with partition ratio $p_{t,m}$ that is offloaded to the m^{th} edge server, its secure transmission energy consumption $E^{\text{tran}}_{t,m}$ can be calculated as

$$E^{\text{tran}}_{t,m} = T^{\text{tran}}_{t,m} P_{u_{t,m}} = \frac{p_{t,m} d_t}{r^{\text{sec}}_{t,m}} P_{u_{t,m}}. \quad (16)$$

The energy consumption of computing the subtasks of Γ_t can be given as

$$E^{\text{comp}}_{t,0} = \chi p_{t,0} c_t f^2_{t,0}, \quad (17)$$

$$E^{\text{comp}}_{t,m} = \chi p_{t,m} c_t f^2_{t,m}, \quad (18)$$

where, $\chi = 10^{-26}$ is the energy coefficient [31].

The energy consumption to complete the m^{th} subtask of the task Γ_t is calculated as

$$E_{t,m} = E^{\text{tran}}_{t,m} + E^{\text{comp}}_{t,m}. \quad (19)$$

For task Γ_t, the total energy consumption E_t can be calculated as the sum of energy consumption of all subtasks as

$$E_t = E^{\text{comp}}_{t,0} + \sum_{m=1}^{M} E_{t,m}. \quad (20)$$

For clarity, all frequently used symbols are summarized in Table I.

D. Problem Formulation

The long-term utility of the computation offloading process in MEC scenarios is considered in this work, and our objective is to maximize the number tasks that are completed before deadlines, while minimizing the total energy consumption and tasks completion time under secure transmission constraints in the long run.

TABLE I
SUMMARY OF NOTATIONS

Symbol	Description
u_t	The IoT user at time slot t
s_m	The m^{th} edge server
Q_0, Q_m	Number of tasks in task queue of IoT user and m^{th} edge server
f_0^{max}, f_m^{max}	Maximum CPU frequencies of IoT user and m^{th} edge server
Γ_t	Task generated at time slot t
d_t	Task data size
c_t	Requested CPU cycles to process task Γ_t
T_t^{max}	Deadline of task Γ_t
$p_{t,0}$, $p_{t,m}$	Partition ratios of subtasks processed by IoT user and m^{th} edge server
$r_{t,m}^{\text{sec}}$	Secrecy transmission rate
$T_{t,m}^{\text{tran}}$	Secure transmission time for offloading m^{th} subtask of task Γ_t
$P_{u_{t,m}}$, $P_{J_{t,m}}$	Transmission power of IoT user and selected friendly jammer
h_{u,s_m}, $h_{J_{t,m},s_m}$, $h_{u,e}$, $h_{J_{t,m},e}$	Small-scale fading
L_{u,s_m}, $L_{J_{t,m},s_m}$, $L_{u,e}$, $L_{J_{t,m},e}$	Path loss
$T_{t,0}^{\text{res}}$, $T_{t,m}^{\text{res}}$	Residual running time of the subtask which is currently being processed
$T_{t,0}^{\text{que}}$, $T_{t,m}^{\text{que}}$	Queuing time
$T_{t,0}^{\text{comp}}$, $T_{t,m}^{\text{comp}}$	Required computing time
$E_{t,m}^{\text{tran}}$	Secure transmission energy consumption for offloading m^{th} subtask of task Γ_t
$E_{t,0}^{\text{comp}}$, $E_{t,m}^{\text{comp}}$	Energy consumption of computing the subtasks

The optimization problem is mathematically formulated as:

$$\begin{aligned}
\max_{\{a_t,\forall t\}\in\pi} \quad & \sum_{t}^{T} \alpha_1\beta_1\nu_t - \alpha_2\beta_2 E_t - \alpha_3\beta_3 T_t \\
\text{s.t.} \quad & a_t = \{\mathbf{P}_t, \mathbf{J}_t, \mathbf{F}_t\} \\
& \mathbf{P}_t = \{p_{t,0}, p_{t,1}, \ldots, p_{t,m}, \ldots, p_{t,M}\} \\
& \mathbf{J}_t = \{J_{t,1}, \ldots, J_{t,m}, \ldots, J_{t,M}\} \\
& \mathbf{F}_t = \{f_{t,0}, f_{t,1}, \ldots, f_{t,m}, \ldots, f_{t,M}\} \\
& p_{t,0} + \sum_{m=1}^{M} p_{t,m} = 1, 0 \le p_{t,0}, p_{t,m} \le 1 \\
& J_{t,m} \in \mathcal{K} = \{0, 1, 2, \ldots, K\} \\
& 0 \le f_{t,0} \le f_0^{\max} \\
& 0 \le f_{t,m} \le f_m^{\max}. \qquad (21)
\end{aligned}$$

where π are optimal policies; β_1, β_2 and β_3 are normalization terms that convert ν_t, E_t and T_t into the same scale; and user preferences are specified by $\alpha_1 + \alpha_2 + \alpha_3 = 1$ with respect to the number of completed tasks, energy consumption and latency; $f_u^{\max}$ and $f_m^{\max}$ are maximum CPU frequencies.

III. PROPOSED METHODOLOGY

In this section, we present a novel DRL approach that integrates VAE, OU process, and Dirichlet distribution into the action network to handle the hybrid and constrained action space in the context of secure computation offloading in MEC systems. We first reformulate the above problem as a MDP, and then we present the details of the proposed DRL approach.

A. Problem Reformulation as MDP

To solve the formulated problem with DRL, we first need to define the elements of DRL for secure MEC. In the MEC network environment in the given time slot t, the state, action, and reward of the MDP are defined as follows.

1) State Space: At time slot t, the state incorporates six components and is denoted as Γ, U, S, r^{sec}, h, L, where, Γ denotes arrived tasks, U and S denote the status of local IoT user and MEC servers, r^{sec} is the secure transmission rate, and h and L are small-scale fading and path loss of channels, respectively.

2) Action Space: The action includes task partition ratio, jammer selection for each subtask, and CPU allocation at each edge server. An action for m^{th} subtask of task Γ_t can be denoted as $a_{t,m} = \{p_{t,m}, J_{t,m}, f_{t,m}\}$, which specifies how each subtask can be local processed or offloaded. Particularly, $p_{t,m}$ represents continuous task partition ratio, where $p_{t,m} \in [0, 1], \forall t, m$. The task partition ratios corresponding to task Γ_t should satisfy the constraint $p_{t,0} + \sum_{m=1}^{M} p_{t,m} = 1$. $J_{t,m}$ represents the index of the selected jammer for the subtask, where $J_{t,m} \in \mathcal{K} = \{0, 1, 2, \ldots, K\}$ when we have multiple jammer candidates for multiple subtasks. $f_{t,m} > 0$ is the continuous CPU frequency allocation, where $0 \le f_{t,0} \le f_u^{\max}$ and $0 \le f_{t,m} \le f_m^{\max}$.

3) Reward Function: In this work, each action contains three decision variables. For each subtask of each task, the partition ratio, selected jammer, and recommended CPU frequency at the local user or each edge server are defined. The policy π maps the observed states to actions. Based on current state s_t, the agent takes an action a_t, and receives a corresponding immediate reward R_t as the feedback:

$$R_t(s_t, a_t) = \alpha_1\beta_1\nu_t - \alpha_2\beta_2 \log_2(E_t) - \alpha_3\beta_3 \log_2 T_t + \mathcal{C}, \qquad (22)$$

where the use of logarithm for $\log_2(E_t)$ applies nonlinear penalization to ensure numerical stability and capture realistic system behavior under energy-performance tradeoffs. A positive constant number $\mathcal{C}$ is used as an incentive for the stability of edge servers.

This immediate reward reflects our multiple objectives, including maximizing the number of completed tasks, and saving energy consumption, while satisfying the security requirement. A weighted sum method is utilized, where different weights are assigned to the number of completed tasks and energy consumption.

4) Optimization Goal: The goal is to find the policy π, that maps current state s_t to the optimal action $a_t, \forall t \in \{1, 2, \ldots T\}$,

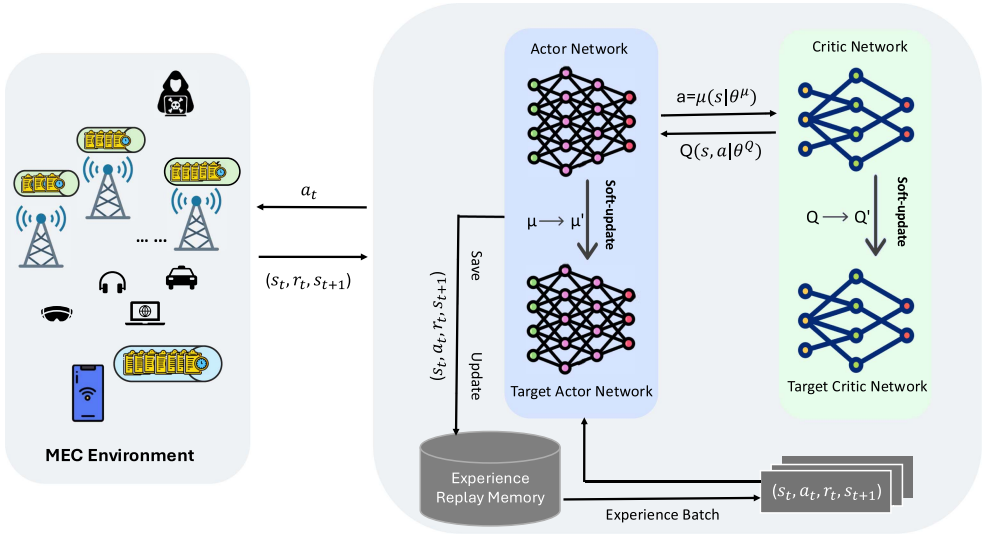

Fig. 2. The framework of ODDPG approach.

which maximizes the expected long-term accumulated immediate rewards, given by

$$\max_{\pi} \mathbb{E}\left[\sum_{t}^{T} R_t(s_t, a_t)\right]. \tag{23}$$

5) Transition Probability: For standard DRL, $p(s'|s,a)$ presents transition probability and is formed as follows:

$$p(s'|s,a) \doteq \Pr\{s_{t+1} = s' | s_t = s, a_t = a\}. \tag{24}$$

It is challenging to model the transition function in such a complex and dynamic MEC environment. Therefore, the model-free DRL technique is adopted in this paper. In addition, to address the complexity of the action space of the formulated problem, we design a novel DRL approach.

B. Proposed DRL Approach

In this paper, we integrate VAE and Dirichlet distribution into DDPG to handle all possible types of action, which can be considered as omni-action, including continuous action space for CPU/GPU frequency control, large discrete space for jammer selection, and the constrained continuous space for task partitioning. In this regard, we refer to the proposed DRL approach as Omni-DDPG (ODDPG).

The proposed ODDPG approach, as shown in Fig. 2, incorporates an MEC environment and an ODDPG agent that learns an optimal policy for decision-making to optimize the expected cumulative reward over time. The decision-making policy of the ODDPG agent incorporates elements from actor-critic methods, target networks, and experience replay. The agent alternates between updating the actor network and critic network: At each time slot, the agent observes the current state; selects an action using the actor network with exploration noise; receives a reward from the environment; and transits to the new state. The training data sets are generated through continuous interaction between the agent and the environment. The agent updates the critic network (Q-value function) and actor network (policy gradient). The target critic and actor networks are also updated periodically by using experiences sampled from the replay buffer iteratively for training stability.

In the following, we introduce the details of the actor network, the critic network, and the target networks. After that, we present the loss function and training process for the networks.

1) Actor Network: The actor network is designed to approximate the policy $\pi(s_t)$, mapping the state s_t to the action a_t at time slot t, given by

$$a_t = \pi\left(s_t|\theta^\mu\right), \tag{25}$$

where θ^μ is the parameters (weights) of the actor network.

Specifically, an action consists of three components, as mentioned above. For task partitioning and offloading, we adopt the Dirichlet distribution to handle the constrained continuous actions [24]. As one vector of each action, $\mathbf{P}_t = \{p_{t,0}, p_{t,1}, \ldots, p_{t,m}, \ldots, p_{t,M}\}$ is continuous for partitioning tasks into smaller subtasks for their corresponding edge servers and should satisfy the constraint $p_{t,0} + \sum_{m=1}^{M} p_{t,m} = 1$, where $p_{t,0}, p_{t,m} \in [0,1]$. Directly letting the actor network output the individual task partition ratios may lead to the violation of the constraint. To this end, we develop action sampling based on a Dirichlet distribution, since it is defined over probability vectors whose components are strictly non-negative and sum to one.

The Dirichlet distribution is defined over a vector $\mathbf{P}_t = \{p_{t,0}, p_{t,1}, \ldots, p_{t,m}, \ldots, p_{t,M}\}$, where each $p_{t,0}, p_{t,m} \in [0,1]$ and $p_{t,0} + \sum_{m=1}^{M} p_{t,m} = 1$. The probability density function is:

$$p\left(\mathbf{P}_t; \boldsymbol{\theta}_t\right) = \frac{1}{B(\boldsymbol{\theta}_t)} \left(p_{t,0}^{\theta_{t,0}-1} \prod_{m=1}^{M} p_{t,m}^{\theta_{t,m}-1}\right), \tag{26}$$

where $\boldsymbol{\theta}_t = \{\theta_{t,0}, \ldots, \theta_{t,m}, \cdots\}$ are concentration parameters, and $B(\boldsymbol{\theta}_t)$ is the normalization term. Further, we can present the

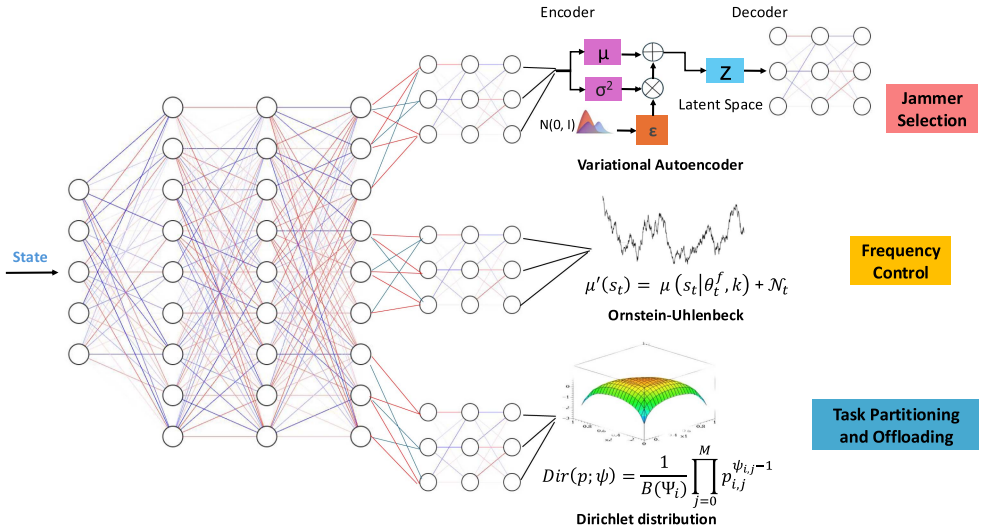

Fig. 3. Actor network.

function in the exponential family form:

$$p_F(x;\theta) = \exp\left(\langle t(x), \theta\rangle - F(\theta) + k(x)\right), \tag{27}$$

where x and θ are the simplified terms of $\mathbf{P}_t$ and $\boldsymbol{\theta}_t$; $t(x)$ is the sufficient statistic which defines a function of x that captures all the information needed to compute the likelihood of θ. $F(\theta)$ is the he log-normalizer that ensures that the distribution integrates (or sums) to 1; $k(x)$ is the carrier measure that adjusts for the base measure, often absorbed or ignored depending on the context. As a result, the probability density function can be simplified and implemented by:

$$p(x;\theta) = \exp\left(\theta_0 \log x_0 + \sum_{m=1}^{M} \theta_m \log x_m - \log B(\theta+1)\right). \tag{28}$$

Although the OU process has been introduced in DDPG [32] to handle continuous action spaces, TD3 [33] shows that continuous actions can simply be generated directly by the actor network, which maps a given state $f_t \in \mathbb{R}^n$ through a function $\pi(s_t|\theta^\pi)$. During training, exploration is encouraged by adding uncorrelated Gaussian noise to the output:

$$a_t = \pi(s_t|\theta^\pi) + \epsilon, \epsilon \sim \mathcal{N}\left(0, \sigma^2\right), \tag{29}$$

where $\epsilon \sim \mathcal{N}(0, \sigma^2)$ noisy action is then clipped to ensure it remains within the valid action bounds of the environment. Note that we have an actor network with its outputs split into three portions, which are assigned to OU for frequency control, VAE for jammer selection, and Dirichlet for task partitioning.

Next, a VAE network is employed to take part in the outputs of the actor network and select jammers for each of the subtasks. Specifically, the general expression can be given:

$$k_1, \ldots, k_m, \cdots = \underset{k_m \in \mathcal{K}}{\operatorname{argmax}} Q\left(\phi\left(s_t\right), k; \theta\right), \tag{30}$$

where $\mathcal{K}$ is the discrete set of available jammers; and $k_m \in \mathcal{K}$ represents the index of the jammer selected for a subtask. The $\phi(s_t)$ is the transition function, and θ is the parameter of the model. Specifically, the conditional VAE is used to construct a compact and decodable latent representation for discrete-continuous hybrid actions [34]. The inputs to the VAE encoder are the state s, the discrete action embedding $e_{\zeta,k}$, and the continuous parameter x_k. The encoder $q_\phi(z \mid x_k, s, e_{\zeta,k})$ generates a Gaussian distribution parameterized by mean μ_x and standard deviation σ_x, from which a latent vector $z_x \in \mathbb{R}^{d_2}$ is sampled. The decoder $p_\psi(x_k \mid z_x, s, e_{\zeta,k})$ reconstructs the continuous parameter x_k, including the $p(\mathbf{P}_t; \boldsymbol{\theta}_t)$ and $\mathcal{N}_t$ as mentioned in the above. Discrete actions are not chosen directly by the VAE but are facilitated through a latent discrete vector $e \in \mathbb{R}^{d_1}$, output by the actor network. The discrete action k is selected by nearest-neighbor lookup in the embedding table: $k = \arg\min_{k' \in K} |e_{\zeta,k'} - e|^2$. Together, the VAE and embedding table enable the actor to operate in a smooth latent space by outputting (e, z_x), which is decoded into the hybrid action (k, x_k).

As shown in Fig. 3, the results of VAE, OU process-like, and Dirichlet distribution are concatenated as a complete action:

$$a_t = \Big\{ Dir(\theta_t^p|k), \mu\big(s_t \mid \theta_t^f, k\big) + \mathcal{N}_t|k, \operatorname{argmax}_\theta Q(\phi(s_t), k \in \mathcal{K}; \theta) \Big\}. \tag{31}$$

2) Critic Network: The critic network evaluates the actions generated by the actor network, which guides the actor network to adjust the parameter values and optimize the decision-making policy. A typical evaluation metric for a state-action pair (s_t, a_t) is the Q value, which is defined as

$$Q(s_t, a_t) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r(s_{t+k}, a_{t+k}) | s_t, a_t\right], \tag{32}$$

where $r(s_{t+k}, a_{t+k})$ is the immediate reward after taking action a_{t+k} in state s_{t+k}, and γ^k indicates the reward is discounted to the k-th power, representing the diminishing importance of future rewards.

The critic network, parameterized by θ, is used to approximate the Q-values as $Q_\theta(s_t, a_t)$.

Target networks: The target actor network $\mu'(s|\theta^{\mu'})$ and target critic network $Q'(s, a|\theta^{Q'})$ are lagged replica of the actor and critic networks. They are used to compute target Q-values for jointly evaluating the actor and critic networks in action generation and Q-value approximation. Specifically, after taking action a_t and observing the state transition from s_t to s_{t+1}, the critic network calculates the target Q-value for the state-action pair (s_t, a_t). The target Q-value y_t, incorporating the immediate reward $r(s_t, a_t)$ and the discounted future reward $\gamma(s_{t+1}, \mu(s_t + 1)|\theta^{Q'})$, is given by:

$$y_t = r(s_t, a_t) + \gamma \left(s_{t+1}, \mu(s_t + 1)|\theta^{Q'} \right), \tag{33}$$

where γ is the discounted future reward.

The target Q-values are collected in the experience replay buffer, and used to construct the loss function for the update of the actor and critic networks. The target networks are updated slowly by using soft parameter updates over time, ensuring the stability in training the networks.

3) Loss Function: A loss function integrates three components: $L_{Dir}(\theta^Q)$, $L_{OU}(\theta^Q)$, and $L_{VAE}(\theta^Q)$ are designed to optimize the DRL model with respect the three types of actions spaces. The total loss function can be given by:

$$L(\theta^Q) = L_{OU}\left(\theta^Q\right) + L_{VAE}\left(\theta^Q\right) + L_{Dir}\left(\theta^Q\right). \tag{34}$$

$L_{OU}(\theta^Q)$ is a regular loss function for the ODDPG approach. It evaluates the temporal difference (TD) error between the predicted Q-value $Q(s_t, a_t|\theta^Q)$ and the target Q-value y_t:

$$L_{OU}\left(\theta^Q\right) = \mathbb{E}_{s_t\sim\rho^\beta, a_t\sim\beta, r_t\sim E}\left[\left(Q\left(s_t, a_t|\theta^Q\right) - y_t\right)^2\right]. \tag{35}$$

where $s_t \sim \rho^\beta, a_t \sim \beta$ and $r_t \sim E$ denote specific state s_t, action a_t, and reward r_t at time step t within state space ρ^β, action space β and expected reward distribution E.

$L_{VAE}(\theta^Q)$ is used for training the VAE, which is a downstream component of the action network, as mentioned above. The model learns to reduce the gap between the inputs x_m and outputs $\tilde{x}_m$ of the VAE by minimizing $\|x_m - \tilde{x}_m\|_2^2$. The loss functions also regularized by a Kullback-Leibler (KL) divergence term $D_{KL}(q_p(\cdot|x_k, s, e_{\zeta,k}, k)|\mathcal{N}(0, I))$, which enforces a structured latent space by regularizing the posterior distribution against a predefined normal distribution:

$$\begin{aligned} L_{VAE}\left(\theta^Q\right) = \mathbb{E}_{s_t\sim\rho^\beta, a_t\sim J, r_t\sim E, z\sim q_p}\left[\|x_m - \tilde{x}_m\|_2^2\right. \\ \left. + D_{KL}\left(q_p(\cdot|x_k, s, e_{\zeta,k}, k)|\mathcal{N}(0, I)\right)\right], \end{aligned} \tag{36}$$

where $\cdot$ denotes a placeholder for the latent variable z.

Finally, $L_{Dir}(\theta^Q)$ is the loss component that incentivizes generating sequences of actions that derive optimal Dirichlet distribution for the task partitioning and offloading, according to the current available resources in edge servers in a security-aware manner:

$$L_{Dir}\left(\theta^Q\right) = \mathbb{E}_{s_t\sim\rho^\beta, a_t\sim p, r_t\sim E}\left[D_{KL}\left(q_p\left(\cdot|Dir(p), k\right)\right)\right]. \tag{37}$$

4) Training Process: In this hybrid-action reinforcement learning framework, the actor is responsible for learning a policy over multiple latent representation branches, each corresponding to different types of action components including discrete, continuous, and distribution-based actions. Given a state s, the actor network parameterized by θ outputs a tuple of latent action representations:

$$\pi_\theta(s) = \left(z_{\text{VAE}}, z_{\text{OU}}, p\right),$$

where $z_{\text{VAE}} \in \mathbb{R}^{d_1}$ is the latent code from a variational autoencoder (VAE) for discrete action selection (e.g., jammer selection); $z_{\text{OU}} \in \mathbb{R}^{d_2}$ is generated via an OU process for modeling temporally correlated continuous actions (e.g., frequency control); and - $p \in \Delta^M$ is a probability vector drawn from a Dirichlet distribution for structured multi-task actions (e.g., task partitioning and offloading).

The latent representations are decoded into original hybrid actions using domain-specific decoding heads: Discrete action k is selected by decoding z_{VAE} via a nearest-neighbor lookup from an embedding table E_ζ; continuous control x_k is decoded from z_{OU} via a transformation network; task allocation probabilities p are directly used from the Dirichlet branch output.

The actor network is trained using the deterministic policy gradient method. In training the actor network, the objective is to maximize the Q-value estimated by the critic network for the selected latent actions:

$$L(\theta) = -\mathbb{E}_{s\sim\mathcal{D}}\left[Q\left(s, \pi_\theta(s)\right)\right] = -\mathbb{E}_{s\sim\mathcal{D}}\left[Q\left(s, z_{\text{VAE}}, z_{\text{OU}}, p\right)\right]. \tag{38}$$

The actor parameters are updated according to the deterministic policy gradient:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim\mathcal{D}}\left[\nabla_a Q(s, a)\big|_{a=\pi_\theta(s)} \cdot \nabla_\theta \pi_\theta(s)\right]. \tag{39}$$

This training approach encourages the actor network to learn semantically structured and task-relevant action representations, which are decoded into hybrid actions that maximize expected return. By leveraging domain-specific structures like variational encoding, temporal noise modeling, and Dirichlet sampling, the actor network achieves robust and scalable decision-making across heterogeneous action modalities.

The training process of our ODDPG approach aims to maximize the cumulative reward through a structured three-phase procedure: initialization, data generation, and learning. The actor is optimized using the deterministic policy gradient method, leveraging feedback from the critic network to refine its policy and update its parameters. The primary goal of the actor is to learn an optimal policy that selects actions maximizing long-term returns. Meanwhile, the critic network estimates the expected Q-values and provides guidance to the actor by evaluating the quality of its actions. The training objective is to minimize the discrepancy between predicted Q-values and target Q-values. This is achieved by minimizing the composite loss function, which guides both actor and critic networks toward convergence to an optimal policy and accurate action-value approximations.

- *Initialization:* Replay memory $\mathcal{D}$ is initialized to store agent experiences. The actor network $\mu(s \mid \theta^\mu)$ and critic network $Q(s, a \mid \theta^Q)$ are initialized with random weights.

Target networks Q' and μ' are initialized with weights $\theta^{Q'} \leftarrow \theta^{Q}$ and $\theta^{\mu'} \leftarrow \theta^{\mu}$.

- *Episode Execution:* For each episode (1 to $\mathcal{M}$), the system state is reset and a random noise process $\mathcal{N}$ is initialized for exploration. The initial state S is preprocessed from environment observations: $S \leftarrow \psi(\langle x_1 \rangle)$.
- *Action Selection and Training Data Generation:* For each time step τ, the agent selects an action a_t (shown in (31)) that consists of task partitioning ratio selection from a Dirichlet distribution $Dir(\theta_t^p|k)$, CPU frequency decision from OU process $\mu(s_t \mid \theta_t^f, k) + \mathcal{N}_t|k$, and jammer selection from VAE $\arg\max_\theta Q(\phi(s_t), k \in \mathcal{K}; \theta)$. The environment returns a reward r_t and next state S'. The tuple (S, A, R, S') is stored in $\mathcal{D}$.
- *Learning and Network Update:* A mini-batch of experiences is sampled from $\mathcal{D}$. The target Q-value y_t is computed as:

$$y_t = r_t + \gamma Q_t' \left(s_{t+1}, \mu' \left(s_{t+1} \mid \theta^{\mu'} \right) \right) + E_\zeta(K) \quad (40)$$

The critic is updated by minimizing the loss $L(\theta^Q)$: $\theta_t \leftarrow \min_{\theta_t} L(\theta^Q)$. The actor is updated using the deterministic policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{T} \nabla_a Q(s, a \mid \theta^Q) \cdot \nabla_{\theta^\mu} \mu(s \mid \theta^\mu) \quad (41)$$

The target critic and actor networks are updated by:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}, \quad \theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'} \quad (42)$$

C. Complexity Analysis

To quantify the computational burden of the proposed Omni-DDPG algorithm, let M denote the number of edge servers, K denote the pool of friendly-jammer candidates, $|\mathcal{S}|$ and $|\mathcal{A}|$ denote the lengths of the state and action vectors, B denote the mini-batch size used for stochastic gradient descent, P_a and P_c denote the parameter counts of the actor and critic networks respectively, and D denote the capacity of the replay buffer.

During each gradient step, the dominant computation results from the inference of the two networks. An actor forward-and-backward sweep has a complexity of $\mathcal{O}(BP_a)$, while the critic's evaluation and back-propagation have a complexity of $\mathcal{O}(BP_c)$ [35]. In the same step, the Dirichlet sampler that produces the offloading ratios adds only a complexity of $\mathcal{O}(M)$, and the VAE head that ranks jammers introduces a complexity of $\mathcal{O}(BP_{\text{VAE}})$, where $P_{\text{VAE}} \ll P_a$. Soft target-network updates and OU-noise generation are with a negligible complexity of $\mathcal{O}(P_a + P_c)$. Hence the total complexity of the training per step is $\mathcal{O}(B(P_a + P_c) + M)$, and an episode of T environment interactions is $\mathcal{O}(TB(P_a + P_c) + TM)$.

The memory usage is dominated by the replay buffer, which requires $\mathcal{O}(D(|\mathcal{S}| + |\mathcal{A}| + 1))$ memory for storing state, action, and reward tuples. The number of parameters in the critic and target networks scales linearly with $P_a + P_c$, while intermediate activations for back-propagation demand $\mathcal{O}(B(|\mathcal{S}| + |\mathcal{A}|))$ memory. With the values $D = 10^6$ and $|\mathcal{S}| + |\mathcal{A}| \approx 10^3$ used in our simulations, which is shown in Table I, the buffer alone needs about 4 GB in 32-bit precision, whereas the networks require two orders of magnitude less memory.

TABLE II
PARAMETER SETTING

Task Data Size (bits)	$[5 \times 10^6, 2 \times 10^7]$
Task Computing Size (cycles)	$[8 \times 10^6, 1 \times 10^7]$
Number of IoT Users	$[50, 1000]$
Number of Edge Servers	$[5, 50]$
Server Maximum Frequency (Hz)	$[4 \times 10^9, 8 \times 10^9]$
Batch Size	256
Discount Rate γ	0.9
Learning Rate η	6×10^{-4}
Autoencoder	128, 64, 32; 32, 64, 128
Actor Network	128, 256, 512, 256, 128

In decision-making, only the actor network is implemented. As a result, the latency of a decision scales as $\mathcal{O}(P_a)$ and its memory usage is likewise $\mathcal{O}(P_a)$. For the configuration of the actor network in our experiments, $P_a \approx 1.5 \times 10^5$, which corresponds to roughly 0.6 MB in 32-bit floating point precision, well within the capabilities of a typical edge CPU or lightweight GPU.

Finally, the analysis shows that the computational cost increases linearly with the number of edge servers M (each server adds one Dirichlet component) and linearly with the jammer pool size K (storing K embedding vectors). Quadratic growth appears only if the hidden layers are widened, leaving ample room for scaling to larger deployments without dramatic cost inflation. Overall, training remains tractable on a single GPU, and the lightweight actor network ensures real-time feasibility once deployed at the edge.

By contrast, hybrid schemes that mix DRL with external optimization routines (for example, implementing a convex solver or heuristic search inside the DRL loop) incur an additional complexity of about $O(C_{opt}N_{iter})$ per decision, where C_{opt} represents the solver cost and N_{iter} the iterations to converge. Removing these innerloop solver implementations makes the endtoend OmniDDPG pipeline run markedly faster, consume less energy, and retain a logically straightforward design.

IV. Simulation Results

In this section, we present detailed simulation results to evaluate the generality, scalability and robustness of the performance of the proposed ODDPG model.

The simulation environment is developed using the Python programming language, with NumPy and PyTorch libraries employed for data preprocessing and implementation of DRL models. To assess the effectiveness of ODDPG, we compare it against several baselines, including conventional DRL algorithms such as DDPG and DQN, as well as a greedy algorithm representing traditional heuristic-based approaches. To emulate a realistic MEC environment, we simulate key network entities, including IoT users, computational tasks, MEC servers, a centralized coordinator, and the DRL agent, each modeled as independent processing units. The key system parameters in the simulations are given in Table II. In each simulation, a random number of IoT devices are simulated, and their task offloading

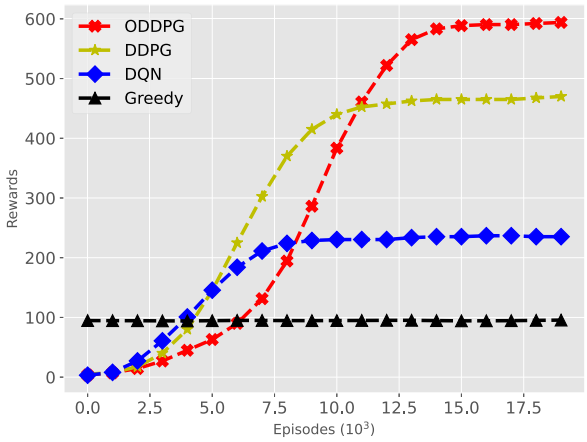

Fig. 4. Rewards.

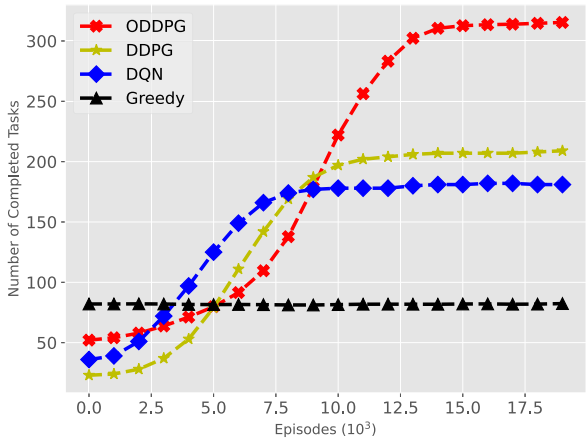

Fig. 5. Completed tasks.

decisions are determined by a shared ODDPG model. In addition, a random number of edge servers are simulated, each with a randomly assigned maximum CPU frequency. Five independent simulations are conducted, and the results are averaged over all tasks of all devices across the simulations under different task offloading approaches.

Fig. 4 illustrates the accumulated rewards achieved by the proposed ODDPG model compared to baseline methods. To support end-to-end learning and address multiple objectives, the reward function comprises three components: the number of successfully completed tasks, energy consumption, and service time cost. Although the proposed ODDPG model requires approximately 600,000 episodes to converge, slightly slower than other models, it ultimately achieves the highest accumulated long-term reward (considering all components jointly, rather than individually). Both ODDPG and DDPG exhibit similar convergence behavior, accumulating more rewards in the later training stages. In contrast, the DQN method discretizes the continuous action space, allowing it to reach an optimal policy with fewer episodes.

All three DRL-based methods underperform the conventional greedy algorithm in the early stages of training due to the exploration process, which involves taking random actions to collect data and learn effective policies. The greedy algorithm, by contrast, does not require training and thus collects higher rewards in the initial episodes. However, in the long term, the proposed ODDPG framework significantly outperforms all baseline approaches in achieving joint optimization in an end-to-end manner.

Furthermore, although the ODDPG model is designed to maximize the expected long-term reward, it is also insightful to analyze the performance of each individual reward component. Fig. 5 presents a comparison of the number of tasks completed before expiration across different methods. Initially, due to random exploration and suboptimal resource allocation, all three DRL-based models complete relatively few tasks. As training progresses and policies are refined through continuous interaction with the MEC environment, task completion rates improve and eventually stabilize. The proposed ODDPG model achieves the highest number of completed tasks before their deadlines, owing to the integration of a Dirichlet distribution and a VAE, which facilitate safe task partitioning under the constraint $p_{t,0} + \sum_{j=1}^{M} p_{t,m} = 1$ with $0 \leq p_{t,0}, p_{t,m} \leq 1$. It is important to note that a completed task is defined as one that satisfies the system's security constraints, specifically, being safely offloaded to the edge server without the risk of eavesdropping, as outlined in the problem formulation. These security constraints are strictly enforced throughout the simulation.

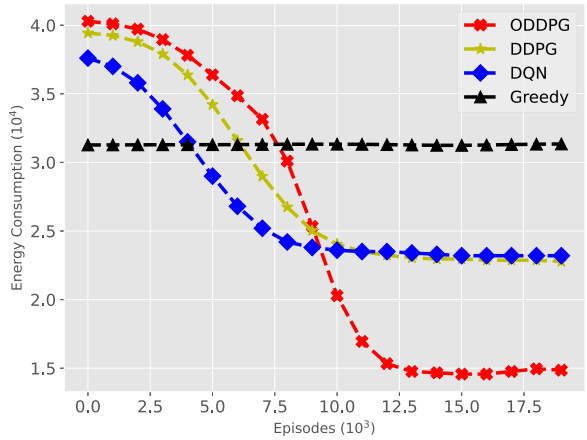

Fig. 6. Energy consumption.

Fig. 6 presents the energy consumption trends of the three DRL-based methods over the course of training. In the early episodes, all models exhibit higher energy consumption due to random policy initialization, the injection of exploratory noise, and the lack of informed decision-making. As training progresses and the models learn from accumulated experience, the frequency of random actions decreases, leading to significant reductions in energy consumption. Although the primary objective of the proposed ODDPG model is to maximize long-term accumulated rewards, it also effectively learns to minimize energy usage over time as a result of improved policy optimization.

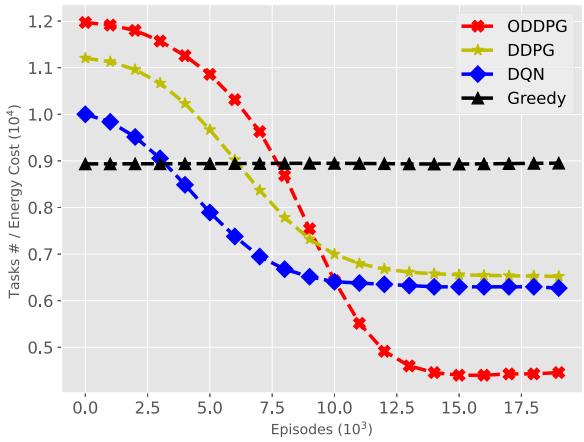

Fig. 7. Ratio of completed tasks to energy cost.

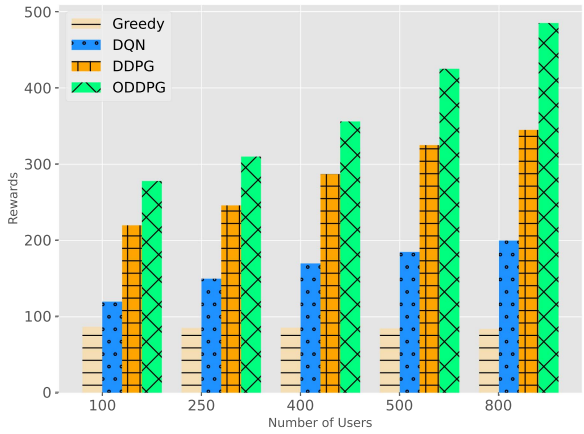

Fig. 8. Reward with respect to the number of users.

Similarly, Fig. 7 depicts the efficiency ratio, defined as the number of completed tasks per unit of energy consumed. While the performance of DDPG and DQN is comparable in terms of task-to-energy efficiency, the proposed ODDPG model consistently achieves a higher number of completed tasks for the same energy expenditure. Notably, all DRL-based methods outperform the conventional greedy algorithm in this metric, highlighting their ability to optimize task processing efficiency in energy-constrained environments.

To evaluate the scalability of the proposed model, we simulate scenarios with an increasing number of source IoT users. Fig. 8 illustrates the accumulated rewards obtained by each method as the number of users grows. All three learning-based approaches, DQN, DDPG, and the proposed ODDPG, consistently outperform the greedy algorithm across varying user counts. Notably, the accumulated rewards for the DRL-based models increase with the number of users, demonstrating their ability to adapt to more complex and dynamic environments. In contrast, the reward obtained by the greedy algorithm remains relatively constant, indicating its limited scalability. Among the DRL methods, ODDPG and DDPG achieve higher performance than DQN, owing to their inherent suitability for continuous action spaces. The proposed ODDPG model, in particular, demonstrates superior scalability, effectively handling a growing number of users while achieving significantly higher long-term rewards.

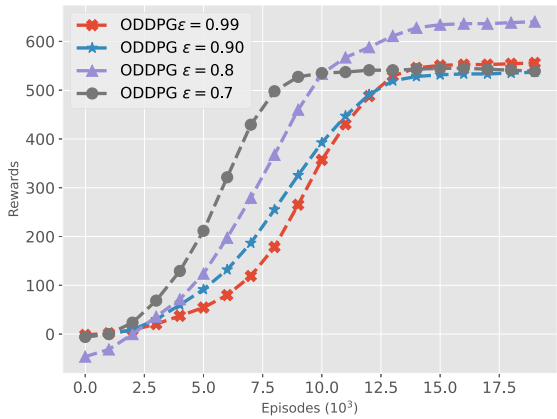

Fig. 9. Proposed model with different exploration rate ϵ.

We also evaluate the proposed model under different exploration configurations, with the results shown in Fig. 9. The exploration rate 1-ϵ directly influences the trade-off between exploration and exploitation. A smaller ϵ typically results in faster convergence due to reduced exploration, but it may lead to suboptimal policies by converging prematurely to local optima. In contrast, a larger ϵ promotes broader exploration, which can delay convergence and increase computational overhead. In our experiments, the model achieved the highest accumulated rewards when $\epsilon = 0.8$, indicating a balanced exploration-exploitation strategy. By comparison, configurations with $\epsilon = 0.99$, 0.90, and 0.70 show similar but lower performance after convergence. Additionally, during early episodes, larger values of ϵ led to lower rewards due to extended exploration periods. These findings suggest that a high exploration rate is not always beneficial. Based on the overall reward performance, we adopt $\epsilon = 0.8$ in our work.

To evaluate the generalizability and efficiency of the proposed ODDPG framework for secure task partitioning and offloading, we illustrate the relationship between the number of IoT users and the corresponding number of generated subtasks across various geographic regions in Fig. 10. As anticipated, the number of subtasks increases with the number of users, demonstrating the framework's ability to dynamically scale in response to user demand. In the scatter plot, different colors denote different regions: densely clustered points (e.g., blue and purple) indicate areas where the model effectively allocates tasks, while more dispersed clusters (e.g., red, yellow, and black) suggest variations influenced by regional disparities in resource availability, security requirements, or offloading strategies.

The accompanying kernel density estimation (KDE) plots provide further insight into the distribution patterns. The top panel shows user distribution across regions, where certain areas, such as the East and Central regions, host higher concentrations of active IoT users. The right panel reveals differences in subtask distribution across regions, suggesting variability in workload intensity and regional computing demand. These observations

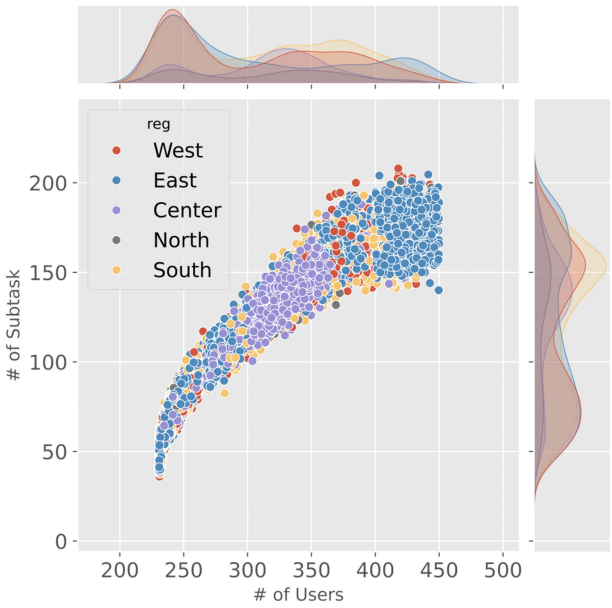

Fig. 10. Relationship between the number of users and generated subtasks across different geographic regions.

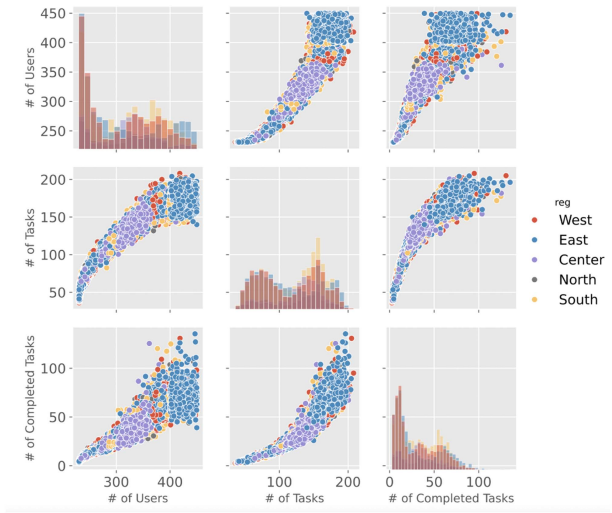

Fig. 11. Relationship among the number of users, generated tasks and completed tasks across different geographic regions.

emphasize the adaptability of the ODDPG model in managing spatial heterogeneity.

Fig. 11 shows a scatter plot matrix that includes three key variables: the number of IoT users, the number of tasks generated, and the number of tasks completed across regions. All pairwise correlations among these variables are positive, indicating that higher user densities generally lead to increased task generation and higher task completion rates. These trends reflect the effectiveness of the ODDPG model in maintaining reliable performance under varying regional conditions. Notably, certain regions exhibit dense clusters with higher task completion efficiency within specific task ranges, likely due to favorable MEC resource availability or more predictable user behavior.

V. Conclusion

In this paper, we have proposed a novel DRL approach, Omni-DDPG, to achieve secure and efficient task offloading in MEC systems under eavesdropping threats. The proposed lightweight and solver-free Omni-DDPG ensures sustained efficiency and scalability, even in large and complex MEC-IoT networks. Beyond secure computation offloading, the proposed Omni-DDPG framework can be readily extended to other network management problems involving hybrid discrete-continuous decision spaces. For the future work, we plan to extend the framework to scenarios with multiple or mobile eavesdroppers, where multiple interception links and time-varying channels are considered. Moreover, to reduce retraining overhead, transfer learning and meta-learning will be explored for faster adaptation of pre-trained agents to new MEC environments with varying network dynamics, and federated learning will be integrated to enhance data privacy and model generalization in distributed MEC-IoT environments.

References

[1] D. C. Nguyen et al., "6G Internet of Things: A comprehensive survey," *IEEE Internet Things J.*, vol. 9, no. 1, pp. 359–383, Jan. 2022.

[2] J. Liu, J. Ren, Y. Zhang, X. Peng, Y. Zhang, and Y. Yang, "Efficient dependent task offloading for multiple applications in MEC-cloud system," *IEEE Trans. Mobile Comput.*, vol. 22, no. 4, pp. 2147–2162, Apr. 2023.

[3] L. Ale, N. Zhang, S. A. King, and D. Chen, "Empowering generative AI through mobile edge computing," *Nature Rev. Elect. Eng.*, vol. 1, no. 7, pp. 478–486, 2024. [Online]. Available: http://dx.doi.org/10.1038/s44287-024-00053-6

[4] M. Li, N. Cheng, J. Gao, Y. Wang, L. Zhao, and X. Shen, "Energy-efficient UAV-assisted mobile edge computing: Resource allocation and trajectory optimization," *IEEE Trans. Veh. Technol.*, vol. 69, no. 3, pp. 3424–3438, Mar. 2020.

[5] M. Li, J. Gao, L. Zhao, and X. Shen, "Deep reinforcement learning for collaborative edge computing in vehicular networks," *IEEE Trans. Cogn. Commun. Netw.*, vol. 6, no. 4, pp. 1122–1135, Dec. 2020.

[6] H. Zhang, X. Liu, Y. Xu, D. Li, C. Yuen, and Q. Xue, "Partial offloading and resource allocation for MEC-assisted vehicular networks," *IEEE Trans. Veh. Technol.*, vol. 73, no. 1, pp. 1276–1288, Jan. 2024.

[7] M. Gao, R. Shen, L. Shi, W. Qi, J. Li, and Y. Li, "Task partitioning and offloading in DNN-task enabled mobile edge computing networks," *IEEE Trans. Mobile Comput.*, vol. 22, no. 4, pp. 2435–2445, Apr. 2023.

[8] J. Ni, X. Lin, and X. Shen, "Toward edge-assisted Internet of Things: From security and efficiency perspectives," *IEEE Netw.*, vol. 33, no. 2, pp. 50–57, Mar./Apr. 2019.

[9] M. Zhao, Z. Wang, K. Guo, R. Zhang, and T. Q. S. Quek, "Against mobile collusive eavesdroppers: Cooperative secure transmission and computation in UAV-assisted MEC networks," *IEEE Trans. Mobile Comput.*, vol. 24, no. 6, pp. 5280–5297, Jun. 2025.

[10] L. Zhang, Y. Sun, Y. Tang, H. Zeng, and Y. Ruan, "Joint offloading decision and resource allocation in MEC-enabled vehicular networks," in *Proc. IEEE 93rd Veh. Technol. Conf.*, 2021, pp. 1–5.

[11] M. Wu, Q. Song, L. Guo, and I. Lee, "Energy-efficient secure computation offloading in wireless powered mobile edge computing systems," *IEEE Trans. Veh. Technol.*, vol. 72, no. 5, pp. 6907–6912, May 2023.

[12] H. Hu, Q. Wang, R. Q. Hu, and H. Zhu, "Mobility-aware offloading and resource allocation in a MEC-enabled IoT network with energy harvesting," *IEEE Internet Things J.*, vol. 8, no. 24, pp. 17541–17556, Dec. 2021.

[13] X. Liu, J. Yu, J. Wang, and Y. Gao, "Resource allocation with edge computing in IoT networks via machine learning," *IEEE Internet Things J.*, vol. 7, no. 4, pp. 3415–3426, Apr. 2020.

[14] L. Huang, X. Feng, A. Feng, Y. Huang, and L. P. Qian, "Distributed deep learning-based offloading for mobile edge computing networks," *Mobile Netw. Appl.*, vol. 27, no. 3, pp. 1123–1130, 2022.

[15] L. Qian, Y. Wu, F. Jiang, N. Yu, W. Lu, and B. Lin, "NOMA assisted multi-task multi-access mobile edge computing via deep reinforcement learning for industrial Internet of Things," *IEEE Trans. Ind. Informat.*, vol. 17, no. 8, pp. 5688–5698, Aug. 2021.

[16] J. Yan, S. Bi, and Y. J. A. Zhang, "Offloading and resource allocation with general task graph in mobile edge computing: A deep reinforcement learning approach," *IEEE Trans. Wireless Commun.*, vol. 19, no. 8, pp. 5404–5419, Aug. 2020.
[17] C. Li et al., "Dynamic offloading for multiuser muti-CAP MEC networks: A deep reinforcement learning approach," *IEEE Trans. Veh. Technol.*, vol. 70, no. 3, pp. 2922–2927, Mar. 2021.
[18] X. Huang, L. He, X. Chen, L. Wang, and F. Li, "Revenue and energy efficiency-driven delay-constrained computing task offloading and resource allocation in a vehicular edge computing network: A deep reinforcement learning approach," *IEEE Internet Things J.*, vol. 9, no. 11, pp. 8852–8868, Jun. 2022.
[19] L. Zhao et al., "MESON: A mobility-aware dependent task offloading scheme for urban vehicular edge computing," *IEEE Trans. Mobile Comput.*, vol. 23, no. 5, pp. 4259–4272, May2024.
[20] C. Sun, X. Wu, X. Li, Q. Fan, J. Wen, and V. C. M. Leung, "Cooperative computation offloading for multi-access edge computing in 6G mobile networks via soft actor critic," *IEEE Trans. Netw. Sci. Eng.*, vol. 11, no. 6, pp. 5601–5614, Nov./Dec. 2024.
[21] A. Gao, S. Zhang, Y. Hu, W. Liang, and S. X. Ng, "Game-combined multi-agent DRL for tasks offloading in wireless powered MEC networks," *IEEE Trans. Veh. Technol.*, vol. 72, no. 7, pp. 9131–9144, Jul. 2023.
[22] T. Song et al., "DRAM: A DRL-based resource allocation scheme for MAR in MEC," *Digit. Commun. Netw.*, vol. 9, no. 3, pp. 723–733, 2023.
[23] W. Wu et al., "Dynamic RAN slicing for service-oriented vehicular networks via constrained learning," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 7, pp. 2076–2089, Jul. 2021.
[24] L. Ale, S. A. King, N. Zhang, A. R. Sattar, and J. Skandaraniyam, "D3PG: Dirichlet DDPG for task partitioning and offloading with constrained hybrid action space in mobile-edge computing," *IEEE Internet Things J.*, vol. 9, no. 19, pp. 19260–19272, Oct. 2022.
[25] A. D. Wyner, "The wire-tap channel," *Bell Syst. Tech. J.*, vol. 54, no. 8, pp. 1355–1387, 1975.
[26] A. Castiglione, M. Nappi, F. Narducci, and C. Pero, "Fostering secure cross-layer collaborative communications by means of covert channels in MEC environments," *Comput. Commun.*, vol. 169, pp. 211–219, 2021.
[27] Y. Wu, G. Ji, T. Wang, L. Qian, B. Lin, and X. Shen, "Non-orthogonal multiple access assisted secure computation offloading via cooperative jamming," *IEEE Trans. Veh. Technol.*, vol. 71, no. 7, pp. 7751–7768, Jul. 2022.
[28] X. He, R. Jin, and H. Dai, "Physical-layer assisted secure offloading in mobile-edge computing," *IEEE Trans. Wireless Commun.*, vol. 19, no. 6, pp. 4054–4066, Jun. 2020.
[29] A. Sanenga, G. A. Mapunda, T. M. L. Jacob, L. Marata, B. Basutli, and J. M. Chuma, "An overview of key technologies in physical layer security," *Entropy*, vol. 22, no. 11, 2020, Art. no. 1261.
[30] K.-L. Besser and E. A. Jorswieck, "Bounds on the secrecy outage probability for dependent fading channels," *IEEE Trans. Commun.*, vol. 69, no. 1, pp. 443–456, Jan. 2021.
[31] S. Guo, J. Liu, Y. Yang, B. Xiao, and Z. Li, "Energy-efficient dynamic computation offloading and cooperative task scheduling in mobile cloud computing," *IEEE Trans. Mobile Comput.*, vol. 18, no. 2, pp. 319–333, Feb. 2019.
[32] T. P. Lillicrap et al., "Continuous control with deep reinforcement learning," US Patent 10,776,692, Sep. 15 2020.
[33] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *Proc. 35th Int. Conf. Mach. Learn.*, 2018, pp. 1587–1596. [Online]. Available: https://proceedings.mlr.press/v80/fujimoto18a.html
[34] B. Li et al., "HyAR: Addressing discrete-continuous action reinforcement learning via hybrid action representation," 2021, *arXiv:2109.05490*.
[35] A. Daniely, N. Srebro, and G. Vardi, "Computational complexity of learning neural networks: Smoothness and degeneracy," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2023, pp. 76272–76297.

Xue Qin (Student Member, IEEE) received the BEng degree from the North University of China (NUC), in 2008, and the MASc degree from the University of Windsor, Windsor, ON, Canada, in 2022. She is currently working toward the PhD degree in electrical and computer engineering with the University of Waterloo, Waterloo, ON, Canada. Her research interests include mobile edge computing, AI, and network security.

Xinyu Huang (Member, IEEE) received the BE degree from Qian Xuesen Experimental Class, Xidian University, Xi'an, China, in 2018, the MS degree in information and communications engineering from Xi'an Jiaotong University, Xi'an, China, in 2021, and the PhD degree in electrical and computer engineering from the University of Waterloo, Waterloo, ON, Canada, in 2025. He is currently a postdoctoral fellow with the Department of Electrical and Computer Engineering, University of Waterloo. His research interests include digital agents, multimedia communications, ISAC, and network management.

Shisheng Hu (Member, IEEE) received the BEng and MASc degrees from the University of Electronic Science and Technology of China (UESTC), Chengdu, China, in 2018 and 2021, respectively, and the PhD degree in electrical and computer engineering from the University of Waterloo, Waterloo, ON, Canada, in 2025. He is currently a postdoctoral fellow with the University of Waterloo. His research interests include edge intelligence, integrated sensing and communications, and machine learning for wireless networks.

Xuemin (Sherman) Shen (Fellow, IEEE) received the PhD degree in electrical engineering from Rutgers University, New Brunswick, NJ, USA, in 1990. He is a University professor with the Department of Electrical and Computer Engineering, University of Waterloo, Canada. His research focuses on network resource management, wireless network security, Internet of Things, 5G and beyond, and vehicular networks. He is a registered professional engineer of Ontario, Canada, an Engineering Institute of Canada fellow, a Canadian Academy of Engineering fellow, a Royal Society of Canada fellow, a Chinese Academy of Engineering foreign member, and an International fellow of the Engineering Academy of Japan, and a distinguished lecturer of the IEEE Vehicular Technology Society and Communications Society. He received "West Lake Friendship Award" from Zhejiang Province in 2023, the President's Excellence in Research from the University of Waterloo in 2022, the Canadian Award for Telecommunications Research from the Canadian Society of Information Theory (CSIT) in 2021, the R.A. Fessenden Award in 2019 from IEEE, Canada, the Award of Merit from the Federation of Chinese Canadian Professionals (Ontario) in 2019, the James Evans Avant Garde Award in 2018 from the IEEE Vehicular Technology Society, the Joseph LoCicero Award in 2015 and Education Award in 2017 from the IEEE Communications Society (ComSoc), the Technical Recognition Award from Wireless Communications Technical Committee in 2019, and the AHSN Technical Committee in 2013. He has also received the Excellent Graduate Supervision Award in 2006 from the University of Waterloo and the Premier's Research Excellence Award in 2003 from the Province of Ontario, Canada. He serves/served as the general chair for the 6G Global Conference'23, and ACM Mobihoc'15, Technical Program Committee chair/co-Chair for IEEE-Globecom'24, 16 and 07, IEEE Infocom'14, IEEE VTC'10 Fall, and the chair for the IEEE ComSoc Technical Committee on Wireless Communications. He is the past president of the IEEE ComSoc, the vice president for Technical & Educational Activities, vice president for Publications, a member-at-large on the Board of Governors, a chair of the Distinguished Lecturer Selection Committee, and a member of IEEE Fellow Selection Committee of the ComSoc. He served as an editor-in-chief of the *IEEE Internet of Things Journal*, *IEEE Network*, and *IET Communications*.