Optimizing the Ratio-Based Offloading in Federated Cloud-Edge Systems: A MADRL Approach

Seifu Birhanu Tadele, Widhi Yahya, Binayak Kar, *Member, IEEE*, Ying-Dar Lin, *Fellow, IEEE*, Yuan-Cheng Lai, and Frezer Guteta Wakgra

***Abstract*—In the evolving landscape of cloud-edge federated systems, Multi-Access Edge Computing (MEC) plays a crucial role by being closer to user equipment (UEs). However, it has limited capacity compared to the cloud, leading to challenges during periods of high network traffic, commonly referred to as hotspot traffic, when MEC resources can become overwhelmed. To mitigate this issue, horizontal and vertical traffic offloading between edges and core sites, and from core sites to the cloud, respectively, is employed. The offloading decisions, crucial for ensuring network efficiency, must be made within seconds. Traditional optimization techniques are unsuitable due to their computational intensity and time-consuming nature, necessitating a shift toward machine learning methods. This research introduces a ratio-based offloading approach, leveraging a multi-agent deep reinforcement learning (MADRL) approach based on the twin-delayed deep deterministic policy gradient (TD3) algorithm. In a comparative evaluation against the simulated annealing (SA) algorithm and single-agent deep reinforcement learning (DRL) approaches, our proposed solution exhibits superior performance, particularly in terms of decision time. The DRL-based approach achieves convergence within seconds, whereas SA takes minutes. Additionally, the average latency experienced by traffic in the multi-agent TD3 configuration is approximately 3–4 times less than in the single-agent configuration.**

***Index Terms*—Cloud computing, DRL, edge computing, offloading, optimization.**

I. Introduction

Cloud computing offers massive centralized computing capacity [1], making it valuable for expanding user equipment (UEs)' computing capabilities due to its rapid elasticity, long-term usability, and suitability for computing-intensive tasks [2], [3]. With the continuous advancement of UEs and communication technologies, a substantial volume of computationally intensive tasks needs to be offloaded from UEs to cloud servers for processing [4], [5]. However, its centralized nature causes it to be distant from UEs, resulting in propagation delay.

The computing paradigm known as Multi-Access Edge Computing (MEC) locates servers close to UEs [6]. This placement provides processing capacity with lower propagation delay compared to centralized cloud computing [7]. MEC can offer distributed and intermediate computing power near UEs, making it suitable for ultra-reliable low-latency communications (URLLC) services within cellular systems. To enhance MEC capacity, research work [8] introduced a two-tier MEC system where computing servers are co-located both with the base station, known as the access network-MEC (AN-MEC), and in the core network-MEC (CN-MEC), hosted at a central office (CO) as Central Office Re-Architected as a Data Center (CORD). The CORD is a new design of a telco central office that replaces closed and proprietary hardware with software running on commodity servers, switches, and access devices [9].

The processing tasks in local UEs often face challenges due to a lack of computing capacity and limited battery power, leading to high computation times and increased latency in UEs' applications [10]. MEC effectively addresses this issue by enabling the offloading of intensive tasks from UEs to MEC servers [11]. However, MEC servers have smaller capacities than cloud servers and may become overloaded by hotspot traffic. Hotspot traffic can occur during events such as music concerts, sporting events, or at transport stations. An overloaded MEC server can offload some traffic to another MEC site [12]. The control plane makes offloading decisions within seconds based on the MEC system's current traffic and system capacity. In contrast, traditional optimization methods can take minutes or hours to converge, failing to meet the control plane's requirement for rapid decision-making [13]. The best offloading decision is challenging to determine due to the presence of multiple distributed sites with varying computing capacities and task arrivals. In such a complex system, offloading optimization must tackle the control plane problem [14], making decisions within seconds to accommodate fluctuating traffic arrivals. A machine learning (ML)-based approach emerges as a convincing solution to the control plane problem, offering the advantage of making quick decisions to accommodate fluctuating incoming

Received 13 October 2023; revised 12 August 2024; accepted 11 November 2024. Date of publication 21 November 2024; date of current version 25 December 2024. This work was supported by the National Science and Technology Council (NSTC), Taiwan under Grant 112-2221-E-011-057 and Grant 111-2221-E-A49-095-MY3. Recommended for acceptance by Dr. Vojislav B. Misic. *(Corresponding author: Widhi Yahya.)*

Seifu Birhanu Tadele, Binayak Kar, and Frezer Guteta Wakgra are with the Department of Computer Science and Information Engineering, National Taiwan University of Science and Technology, Taipei 106, Taiwan (e-mail: seife.brhan@gmail.com; bkar@mail.ntust.edu.tw; haftiyam@gmail.com).

Widhi Yahya is with the Faculty of Computer Science, University of Brawijaya, Malang 65145, Indonesia, and also with the Department of Electrical Engineering and Computer Science, National Yang Ming Chiao Tung University, Hsinchu 300, Taiwan (e-mail: widhi.yahya@ub.ac.id).

Ying-Dar Lin is with the Department of Computer Science, National Yang Ming Chiao Tung University, Hsinchu 300, Taiwan (e-mail: ydlin@cs.nycu.edu.tw).

Yuan-Cheng Lai is with the Department of Information Management, National Taiwan University of Science and Technology, Taipei 106, Taiwan (e-mail: laiyc@cs.ntust.edu.tw).

Digital Object Identifier 10.1109/TNSE.2024.3501398

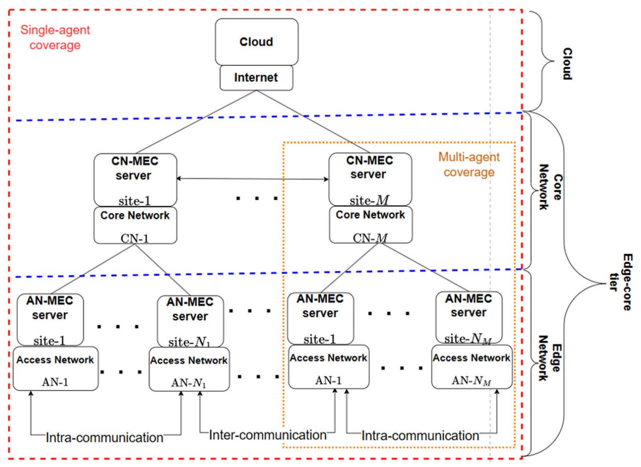


Fig. 1. Three-tier cloud-edge architecture.

traffic. However, establishing data labels for model training in supervised learning is costly due to the dynamic nature of MEC systems [15].

This research investigates the optimization of ratio-based offloading in a three-tier cloud-edge federated system. Fig. 1 illustrates a three-tier cloud-edge computing architecture for efficient network management and service delivery. At the top tier, the cloud connects to the Internet, providing control and computational resources for high-level communication and data processing. The middle tier, the Core Network (CN), features multiple CN-MEC servers distributed across various sites, enabling centralized control and resource management. The bottom tier, the Access Network (AN), includes several AN-MEC servers across multiple sites, offering localized network access and computing resources. In this cloud-edge federated system, ratio-based offloading involves determining both where to offload and how much to offload. In contrast, binary offloading simply decides whether to offload the incoming traffic or not. In a cloud-edge federated system, ratio-based offloading using traditional optimization methods can be time-consuming due to the extensive action space [13].

To address this issue, reinforcement learning (RL) is a valuable technique that can learn an optimal offloading decision by directly interacting with the environment for offloading optimization. It uses trial and error to determine a sub-optimal offloading solution, instead of waiting for the algorithm to converge to the best solution. Hence, we proposed a ratio-based offloading strategy that utilizes multi-agent deep reinforcement learning (MADRL) based on the twin-delayed deep deterministic policy gradient (TD3) algorithm [16]. To the best of our knowledge, this is the first study to propose a MADRL-based TD3 approach for determining the optimal offloading probability in cloud-edge federated systems, considering both vertical and horizontal offloading directions. In this research, we used the terms offloading ratio and offloading probability interchangeably. Our main technical contributions are summarized as follows:

1) We investigated ratio-based offloading optimization, considering horizontal and vertical offloading decisions within control plane systems where fast decision-making is critically important.
2) We designed a cloud-edge RL environment where the agent interacts with it to optimize decisions or actions to evaluate system performance.
3) We proposed a MADRL-based TD3 algorithm to solve ratio-based offloading decisions.
4) We evaluated the performance of the proposed solution by conducting simulations that considered various factors such as decision time, convergence time, decision quality, and unpredictable parameters.

The remaining sections of this research work are organized as follows: In Section II, previous research on ML-based offloading in MEC is presented. In Section III, the system model and the problem formulation are defined. The solution is presented in Section IV. The simulation results and findings are discussed in Section V, while Section VI provides the conclusion of this study.

II. Related Works

In this section, we reviewed previous research on MEC offloading that utilized supervised learning, RL, and DRL approaches, as summarized in Table I. Our analysis focused on both agent and network models. For the agent model, we examine parameters such as the control plane for optimal traffic offloading decisions and the management plane for device-based offloading decisions. We consider online learning for real-time updates, crucial for dynamic MEC environments. We review supervised learning and RL or DRL for their adaptability and efficiency, analyzing both single-agent and multi-agent systems to understand their impact on complexity and scalability, which is critical for offloading problems. Additionally, it assesses whether the studies compare machine learning approaches with traditional optimization techniques, highlighting their advantages and limitations. For the network model, we discuss target networks, objectives, constraints, and unknown parameters, that are essential for understanding the scope, goals, and practical challenges of this study.

Liu et al. [17] addressed offloading and resource allocation with DDoS resistance using a Bayesian Q-learning game and a greedy Q-learning algorithm. However, their method presumes predictable attack patterns, which may not be feasible in real-world situations. Qiu et al. [18] optimized MEC offloading for Blockchain and general computation tasks to minimize costs and enhance algorithm convergence speed using an adaptive genetic algorithm, but their approach may not generalize well to other MEC applications. Khayyat et al. [19] used deep Q-learning with multiple DNNs to reduce latency and energy consumption, demonstrating superior performance over a single DNN, though scalability is a concern due to increased overhead. Qu et al. [20] improved DNN portability in MEC offloading with meta-learning, achieving better performance and convergence times but only considered binary offloading decisions. Wang et al. [21] proposed a dynamic MEC offloading solution using MRL to adapt quickly to multiple tasks, but training these models

TABLE I
RELATED WORKS ON ML-BASED OFFLOADING OPTIMIZATION

Papers	Agent Model						Network Model			
	Problem		Online Learning	Approach	Agent	Machine Learning vs. Traditional Optimization	Target Network	Objective	Constraint	Unknown parameters
	Control Plane	Management Plane								
[17]	X	✓	X	Bayesian Q-Learning	1	X	$D \uparrow E \uparrow C$	Defend against DDoS	Latency	Available Resource
[18]			✓	DRL+GA	1		$D \uparrow E \uparrow C$	Minimize Cost and Convergence time	Resource	Available Resource
[19]	UE			RL	1		$E \uparrow C$	Minimize Latency and Energy	Resource	Available Resource
[20]		X		MRL	1		$D \uparrow E, D \uparrow C$	Minimize Cost and Convergence time	Resources	Environments
[21]				MRL	1		$D \uparrow E$	Minimize Convergence time	Resources	Available Resources
[22]				DRL	1		$D \uparrow E, D \uparrow C$	Minimize Cost	Resources	Available Resources
[23]			X	DL-Supervised	1		$D \uparrow E$	Minimize Energy	Resources	Available Resources
[24]			✓	MAQDRL	n	✓	$D \uparrow E$	Minimize Latency and Energy	Latency	Available Resources
[25]				DRL(DQN)	1		$E \uparrow C$	Minimize Energy	Latency	Arrival Traffic
Ours	Network			MADRL(TD3)	1,n		$D \uparrow E, E \leftrightarrow E, E \uparrow CN, E \uparrow C$	Minimize Latency	Latency	Arrival Traffic & Available Resources

$\uparrow$: **Upward Vertical Offloading;** $\leftrightarrow$: **Bi-directional Horizontal Offloading;** D: **Device;** E: **Edge;** CN: **Core Network;** C: **Cloud**

is resource-intensive. Lu et al. [22] enhanced DQN-based offloading for time-dependent tasks with LSTM layers, improving accuracy but increasing complexity and computation time. Ali et al. [23] developed an energy-efficient DL-based offloading algorithm to minimize cost and energy consumption, requiring well-labeled data that is difficult to obtain in dynamic systems. Wu et al. [24] introduced a lightweight task offloading algorithm for EIIoT using an improved MAPPO algorithm to minimize delay and energy consumption, but did not address performance under varying network conditions. Ale et al. [25] focused on minimizing UE energy consumption with a DRL-based approach, effectively managing energy but it may not adequately address varying task priorities.

Most studies [19], [20], [21], [22], [23], [24], [25] have addressed the control plane problem, requiring UE to handle global information or communicate intensively with a central monitoring system for optimal offloading decisions. In contrast, [17] and [18] focused on management plane problems. However, in this paper, we addressed the control plane problem, where the network handles global information. Furthermore, most prior studies [17], [18], [19], [20], [21], [22], [23] did not compare their methods with traditional optimization techniques. Although studies [24] and [25] included such comparisons, they did so with certain limitations, such as not considering response time in their evaluations. In contrast, this paper considers various parameters, including response time, while comparing the performance of machine learning with traditional optimization techniques. The articles [17] and [23] used Q-learning and supervised learning, respectively, relying on offline methods, whereas most studies including [18], [19], [20], [21], [22], [23], [25] and this paper, employed RL/DRL approaches, which are preferred for MEC offloading. Regarding the number of agents, most of the existing works [17], [18], [19], [20], [21], [22], [23], [25] used a single agent, whereas the article [24] used multi-agents in their approaches. This paper employs both single-agent and multi-agent approaches. Furthermore, all the studies [17], [18], [19], [20], [21], [22], [23], [24], [25] focused on vertical offloading, primarily using binary offloading to determine the offloading location. In contrast, this paper introduces ratio-based offloading optimization for cloud-edge federated systems, considering both horizontal and vertical directions. This analysis highlights our research's unique contributions to performance comparison, addressing control plane problems, utilizing a flexible agent approach, and offloading in both directions.

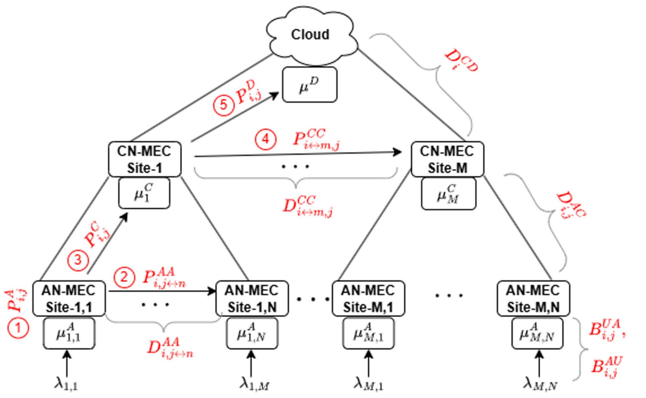


Fig. 2. Cloud-edge ratio-based offloading.

III. SYSTEM MODEL AND PROBLEM FORMULATION

In this section, we present an overview of the cloud-edge federated model with optimal ratio-based offloading and the latency minimization problem. In Table II, we present the notations and descriptions used for latency model construction and problem formulation.

A. Cloud-Edge Federated Model With Ratio-Based Offloading

Fig. 2 illustrates the cloud-edge ratio-based offloading model. Arrival traffic denoted as $\lambda_{i,j}$, is offloaded to the jth AN-MEC site of the ith CN-MEC site, including both normal and hotspot traffic. Each MEC site has servers with capacities μ^D, μ_i^C, and $\mu_{i,j}^A$ for the cloud, CN-MEC site, and AN-MEC site, respectively. There is a set of up to M CN-MEC sites vertically connected to the cloud. Each CN-MEC site is further connected to up to N AN-MEC sites. The set of AN-MEC sites for the ith CN-MEC site is denoted by N_i. MEC sites are interconnected through optical cables. The propagation delays are denoted as follows:

TABLE II
NOTATION AND DESCRIPTION FOR CLOUD-EDGE ARCHITECTURE

Category	Notations	Descriptions	Attribute
Topology	D	cloud site	Input
	$\mathcal{M}$	Set of CN-MEC site, $\mathcal{M} = 1, 2, 3, \cdots, M$	Input
	$\mathcal{N}_i$	Set of AN-MEC sites of ith CN-MEC site, $\mathcal{N}_i = 1, 2, 3, \cdots, N$	Input
	$d^{AC}_{i,j}$	Distance between jth AN-MEC site and ith CN-MEC site	Input
	$d^{AA}_{i,j\leftrightarrow n}$	Distance between jth AN-MEC site and nth AN-MEC site of ith CN-MEC site	Input
	$d^{CC}_{i\leftrightarrow m}$	Distance between ith CN-MEC site and mth CN-MEC site	Input
	d^{CD}_{i}	Distance between ith CN-MEC site and cloud	Input
Capacity	μ^D	Capacity of cloud system	Input
	μ^C_i	Capacity of ith CN-MEC site	Input
	$\mu^A_{i,j}$	Capacity of jth AN-MEC site of ith CN-MEC site	Input
	$B^{UA}_{i,j}$	Uplink bandwidth at jth AN-MEC site of ith CN-MEC site	Input
	$B^{AU}_{i,j}$	Down bandwidth at jth AN-MEC site of ith CN-MEC site	Input
Traffic	$\lambda_{i,j}$	Arrival traffic rate at jth AN-MEC site of ith CN-MEC site	Input
	$\lambda^A_{i,j}$	Traffic that is served at jth AN-MEC site of ith CN-MEC site	Output
	$\lambda^{AA}_{i,n}$	Traffic that is served at $(j-1)$th AN-MEC site of ith CN-MEC site	Output
	λ^C_i	Traffic that is served at ith CN-MEC site	Output
	λ^{CC}_m	Traffic that is served at $(i-1)$th CN-MEC site	Output
	λ^D	Traffic that is served at cloud	Output
	p	Packet size	Input
Latency	$D^{AA}_{i,j\leftrightarrow n}$	Propagation delay between jth AN-MEC site and nth AN-MEC site of ith CN-MEC site	Input
	$D^{CC}_{i\leftrightarrow m}$	Propagation delay between ith CN-MEC site and mth CN-MEC site	Input
	$D^{AC}_{i,j}$	Propagation delay between jth AN-MEC site and ith CN-MEC site	Input
	D^{CD}_{i}	Propagation delay between ith CN-MEC site and cloud	Input
	$l_{i,j}$	Average latency of arrival traffic at jth AN-MEC site of ith CN-MEC site	Output
	l	System average latency	Output
Offloading	$P^A_{i,j}$	Probability of arrival traffic at jth AN-MEC of ith CN-MEC being served at source AN-MEC site	Output
	$P^{AA}_{i,j\rightarrow n}$	Probability of arrival traffic at jth AN-MEC ith CN-MEC being offloaded to another AN-MEC (nth AN-MEC)	Output
	$P^C_{i,j}$	Probability of arrival traffic at jth AN-MEC of ith CN-MEC being served at source CN-MEC site (ith CN-MEC)	Output
	$P^{CC}_{i\rightarrow m,j}$	Probability of arrival traffic at jth AN-MEC ith CN-MEC being offloaded to another CN-MEC (mth CN-MEC)	Output
	$P^D_{i,j}$	Probability of arrival traffic at jth AN-MEC ith CN-MEC being offloaded to cloud	Output

$D^{AC}_{i,j}$ for the delay between the jth AN-MEC and ith CN-MEC, $D^{AA}_{i,j\rightarrow n}$ for the delay between the jth AN-MEC and nth AN-MEC, $D^{CC}_{i\rightarrow m}$ for the delay between the ith CN-MEC and mth CN-MEC, and D^{CD}_{i} for the delay between the ith CN-MEC and the cloud. Fig. 2 shows that upon receiving traffic from UEs, the AN-MEC site can perform five possible offloading directions as follows. 1) Local offloading to the source AN-MEC site from UEs with offloading ratio $P^A_{i,j}$. 2) Horizontal offloading from AN-MEC site to neighboring AN-MEC sites with offloading ratio $P^{AA}_{i,j\rightarrow n}$. 3) Vertical offloading from AN-MEC site to CN-MEC sites with offloading ratio $P^C_{i,j}$. 4) Horizontally from CN-MEC site to neighboring CN-MEC sites with offloading ratio $P^{CC}_{i\rightarrow m,j}$. 5) Vertically from CN-MEC site to the centralized cloud with offloading ratio $P^D_{i,j}$. These offloading ratios are determined by the proposed solution, considering system performance.

B. Latency Calculation

We modeled the system using the M/M/1 queuing model, which features a single server where arrivals follow a Poisson process and service times are exponentially distributed. To optimize system latency, we found that sequentially allocating tasks to AN-MEC, CN-MEC, and the cloud does not yield optimal results. We formulated latency by considering computation, communication, and propagation delays. Propagation delay was calculated by dividing the distance between sites by the speed of light, based on optical cables used for their low attenuation and high bandwidth in high-speed data transmission. For the propagation delay of the router, we assumed the router is located in the central office, connecting AN-MECs to other AN-MECs and CN-MECs to some AN-MECs, as is typical in network architectures. This central placement ensures efficient and reliable connectivity, making delay calculations straightforward for AN-MECs. However, determining the precise number of routers in the cloud is complex due to the abstracted and variable nature of cloud infrastructure. The latency calculations are as follows:

1) Computing Latency by Source AN-MEC: The first direction is the arrival traffic served at source AN-MEC, which is the first entity that receives arrival traffic. The latency of traffic that is served in the source AN-MEC site, denoted as $l^A_{i,j}$, is the sum of radio access network (RAN) uplink time, AN-MEC computing time, and RAN downlink time. This can be calculated as

$$l^A_{i,j} = \frac{1}{\frac{B^{UA}_{i,j}}{p} - \lambda_{i,j}} + \frac{1}{\mu^A_{i,j} - \lambda^A_{i,j}} + \frac{1}{\frac{B^{AU}_{i,j}}{p} - \lambda_{i,j}}, \quad (1)$$

where $\lambda_{i,j}^{A} = P_{i,j}^{A}\lambda_{i,j} + \sum_{n} P_{i,n\to j}^{AA}\lambda_{i,n}, \forall n \in N_i\backslash\{j\}$ is total traffic served at the jth AN-MEC of the ith CN-MEC. This access network receives traffic from UEs at a rate of $\lambda_{i,j}$.

2) Computing Latency by Neighboring AN-MEC: Traffic arriving at an AN-MEC site (belonging to the ith CN-MEC) may overload that site. The traffic can be offloaded to other AN-MEC sites that belong to the same ith CN-MEC. When horizontal offloading is carried out, the offloaded traffic experiences an extra propagation delay between AN-MEC sites. The total latency of offloaded traffic to the nth AN-MEC site, denoted as $l_{i,j\to n}^{AA}$, is equal to the sum of RAN uplink time, nth AN-MEC computing time, RAN downlink time, and back-and-forth propagation delay between the jth and nth AN-MECs sites. This can be expressed as

$$l_{i,j\to n}^{AA} = \frac{1}{\frac{B_{i,j}^{UA}}{p} - \lambda_{i,j}} + \frac{1}{\mu_{i,n}^{A} - \lambda_{i,n}^{AA}} + \frac{1}{\frac{B_{i,j}^{AU}}{p} - \lambda_{i,j}} + 2D_{i,j\leftrightarrow n}^{AA},$$
$$\text{where } \lambda_{i,n}^{AA} = P_{i,n}^{A}\lambda_{i,n} + \sum_{x} P_{i,x\to n}^{AA}\lambda_{i,x}, \forall x \in \mathcal{N}_i\backslash\{n\}, \quad (2)$$

$D_{i,j\leftrightarrow n}^{AA}$ is computed by dividing $d_{i,j\leftrightarrow n}^{AA}$ by the speed of light. At each AN-MEC site, the total arrival traffic equals the sum of vertically offloaded traffic from its AN-MEC and horizontally offloaded traffic from its neighbors.

3) Computing Latency by CN-MEC: Once all AN-MEC sites are overloaded, arriving traffic at the jth AN-MEC of the ith CN-MEC can be vertically offloaded to the CN-MEC sites. The total latency of traffic offloaded to the CN-MEC sites, denoted as $l_{i,j}^{C}$, is the sum of RAN uplink time, CN-MEC processing time, propagation delay between the AN-MEC and CN-MEC sites, and RAN downlink time. This can be expressed as

$$l_{i,j}^{C} = \frac{1}{\frac{B_{i,j}^{UA}}{p} - \lambda_{i,j}} + \frac{1}{\mu_{i}^{C} - \lambda_{i}^{C}} + \frac{1}{\frac{B_{i,j}^{AU}}{p} - \lambda_{i,j}} + 2D_{i,j}^{AC},$$
$$\text{where } \lambda_{i}^{C} = \sum_{n} P_{i,n}^{C}\lambda_{i,n}, \forall n \in \mathcal{N}_i + \sum_{m}\sum_{n} P_{m\to i}^{CC}\lambda_{m,n},$$
$$\forall m \in \mathcal{M}\backslash\{i\}, \forall n \in \mathcal{N}_{\boldsymbol{m}}. \quad (3)$$

$D_{i,j}^{AC}$ is computed by dividing $d_{i,j}^{AC}$ by the speed of light. The total traffic consists of the sum of vertically offloaded traffic from its AN-MEC sites and horizontally offloaded traffic from neighbor CN-MEC sites.

4) Computing Latency by Neighboring CN-MEC: The incoming traffic at the jth AN-MEC of the ith CN-MEC is offloaded to the neighbors of CN-MEC site. This offloading option incurs a higher propagation delay than the preceding one, which was offloaded to CN-MEC because it involves both vertical and horizontal propagation delays between AN-MEC and CN-MEC sites. The total latency of traffic offloaded to mth CN-MEC site, denoted as $l_{i\leftrightarrow m,j}^{CC}$, is the sum of RAN uplink time, CN-MEC computing time, RAN downlink time, back-and-forth propagation delay between jth AN-MEC and ith CN-MEC, back-and-forth propagation delay between ith and mth CN-MECs. This can be calculated as

$$l_{i\leftrightarrow m,j}^{CC} = \frac{1}{\frac{B_{i,j}^{UA}}{p} - \lambda_{i,j}} + \frac{1}{\mu_{m}^{C} - \lambda_{m}^{CC}} + \frac{1}{\frac{B_{i,j}^{AU}}{p} - \lambda_{i,j}}$$
$$+ 2D_{i,j}^{AC} + 2D_{i\leftrightarrow m}^{CC}, \quad (4)$$

where $\lambda_{m}^{CC} = \sum_{j} P_{m,j}^{C}\lambda_{m,j}, \forall j \in \mathcal{N}_m + \sum_{x}\sum_{n} P_{x\to m,n}^{CC}\lambda_{x,n}, \forall x \in \mathcal{M}, \forall n \in \mathcal{N}_x$, is the total traffic executed by mth core network. $D_{i\leftrightarrow m}^{CC}$ is computed by dividing $d_{i\leftrightarrow m}^{CC}$ by the speed of light.

5) Computing Latency by Cloud: A heavy traffic arrival can overload all edge sites, leading to increased computing latency, which requires offloading to the cloud. The traffic offloaded to the cloud experiences a propagation delay from AN-MEC to CN-MEC and from CN-MEC to the cloud. The total latency of cloud traffic, denoted as $l_{i,j}^{D}$, is determined by the sum of RAN uplink time, AN-MEC computing time, RAN downlink time, twice the propagation delay between the jth AN-MEC and the ith CN-MEC, and twice the propagation delay to the cloud. This can be calculated as

$$l_{i,j}^{D} = \frac{1}{\frac{B_{i,j}^{UA}}{p} - \lambda_{i,j}} + \frac{1}{\mu^{D} - \lambda^{D}}$$
$$+ \frac{1}{\frac{B_{i,j}^{AU}}{p} - \lambda_{i,j}} + 2D_{i,j}^{AC} + 2D_{i}^{CD}, \quad (5)$$

where $\lambda^{D} = \sum_{i}^{M}\sum_{j}^{N_i} P_{i,j}^{D}\lambda_{i,j}$, is the total traffic executed by the cloud. D_{i}^{CD} is computed by dividing d_{i}^{CD} by the speed of light.

C. Problem Formulation

Latency models for all possible traffic destinations in the cloud-edge system are provided in (1), (2), (3), (4), and (5). As illustrated in Fig. 2, we assume that traffic arrives at AN-MEC sites before being offloaded to another site. The average latency of incoming traffic sourced from the jth AN-MEC site of the ith CN-MEC site ($l_{i,j}$) is calculated as We have considered three-tier cloud-edge federated systems, which consist of multiple AN-MEC sites, CN-MEC sites, and the cloud. Our goal is to minimize the average system latency (l) within these cloud-edge federated systems. By utilizing the analytical expressions provided in (6) shown at the bottom of this page, we can calculate the average system latency l as

$$l = \frac{\sum_{i}\sum_{j} l_{i,j}}{\sum_{i}\sum_{j}\lambda_{i,j}}, \quad (7)$$

$$l_{i,j} = \frac{\left(\begin{array}{c} l_{i,j}^{A}\left(P_{i,j}^{A}\times\lambda_{i,j}\right) + \sum_{n\in N_i\backslash\{j\}} l_{i,j\to n}^{AA}\left(P_{i,j\to n}^{AA}\times\lambda_{i,j}\right) + l_{i,j}^{C}\left(P_{i,j}^{C}\times\lambda_{i,j}\right) \\ + \sum_{m\in\mathcal{M}\backslash\{i\}} l_{i\to m,j}^{CC}\left(P_{i\to m,j}^{CC}\times\lambda_{i,j}\right) + l_{i,j}^{D}\left(P_{i,j}^{D}\times\lambda_{i,j}\right)\end{array}\right)}{\lambda_{i,j}}. \quad (6)$$

where $l_{i,j}$ the average latency of incoming traffic sourced from the jth AN-MEC site of the ith CN-MEC site given in (6). Thus, the objective function of the problem with the given constraints is stated as

$$\text{Minimize}(l), \tag{8}$$

subject to,

$$P^{A}_{i,j} + P^{AA}_{i,j\to n}, \forall n \in \mathcal{N}_i\backslash\{\text{j}\} + P^{C}_{i,j}$$
$$+ P^{CC}_{i\to m,j}, \forall m \in \mathcal{M}\backslash\{i\} + P^{D}_{i,j} = 1, \tag{9a}$$
$$\left\{0 \le P^{A}_{i,j}, P^{AA}_{i,j\to n}, P^{C}_{i,j}, P^{CC}_{i\to m,j}, P^{D}_{i,j} \le 1\right\}, \tag{9b}$$
$$\lambda^{A}_{i,j} + \lambda^{AA}_{i,n} + \lambda^{C}_{i} + \lambda^{CC}_{m} + \lambda^{D} \le \lambda_{i,j}, \tag{9c}$$
$$\lambda_{i,j} \ge 0 \quad \forall i \in \mathcal{M}, \forall j \in \mathcal{N}_i. \tag{9d}$$

The constraints in (9a) guarantee that the summation of the offloading ratio must be equal to one. The constraints in (9b) ensure that the value of each offloading ratio should be between zero and one. Equation (9c) ensures that the sum of all offloaded traffic must be less than or equal to the total arrival traffic. The constraint in (9d) guarantees the non-negativity of each traffic arrival rate.

IV. Proposed Solution

The offloading function of the control plane module determines the cloud-edge offloading ratio. This function, known as an agent in DRL, interacts with the edge-cloud system to find the optimal offloading ratio. The agent can be placed in distributed central offices (CN-MEC) to facilitate traffic redirection to edge servers. We have used a modified DRL-based TD3 algorithm, which is a model-free, online, off-policy DRL method. A TD3 agent is an actor-critic DRL agent that searches for an optimal policy that maximizes the expected cumulative long-term reward [26]. TD3 agents can be trained in environments with continuous or discrete observation and continuous action spaces. This study investigates the impact of single and multi-DRL agents along SA optimization on offloading. The primary distinction between single and multi-agents lies in the agent's coverage. With a large amount of input and actions, the single agent covers all cloud-edge systems. Conversely, the use of multi-agents reduces coverage, resulting in a smaller input and action space.

The agent is trained using past experiences stored in a memory buffer. It stores experiences in the format of states, actions, next states, and rewards (s, a, s', r). The agent learns the optimal action to take given a particular state.

A. Ratio-Based Cloud-Edge RL Environment

The environment represents the external system or process with which an RL agent interacts. We have designed a cloud-edge environment that involves task control problems, requiring appropriate decisions to be made at each moment in time to achieve their objectives. We have implemented RL-based agents to interact with the environment and make quick decisions. We define the states, actions, and rewards for cloud-edge offloading optimization in the designed environment as follows.

1) State: It is a representation of the current environment that the agent can observe and includes all relevant information necessary for decision-making. It constitutes the cloud-edge site's information, arrival traffic rates ($\lambda_{i,j}$), computation resources ($\mu^{D}, \mu^{C}_{i}, \mu^{A}_{i,j}$), and site latency ($l_{i,j}$). The following are state arrays built from states that are used as inputs to policy and Q-value networks,

$$T_t = \{\lambda_{i,j}\}^{M,N_i}_{i=1,j=1}, \tag{10}$$
$$C_t = \left\{\mu^{C}_{i}\right\}^{M}_{i=1}, \left\{\mu^{A}_{i,j}\right\}^{M,N_i}_{i=1,j=1}, \left\{\mu^{D}\right\}, \tag{11}$$
$$\text{and } L_t = \{l_{i,j}\}^{M,N_i}_{i=1,j=1}, \tag{12}$$

where T_t denotes a set of all AN-MEC sites' arrival traffic rates at the tth step. C_t is an array containing the computing capacity available at the AN-MEC, CN-MEC, and cloud. Capacity changes throughout time in response to the rate of arrival traffic. L_t composes the latency of all sites. T_t, C_t, and L_t are composed as an environment state s_t at tth step,

$$s_t = [T_t, C_t, L_t]. \tag{13}$$

2) Action: An action in a cloud-edge environment is a set of offloading ratios that determine the amount of offloaded traffic to the destinations. These offloading ratios can be represented as follows:

- $P^{A}_{i,j}$ is the offloading ratio of traffic computed in source AN-MEC.
- $P^{AA}_{i,j\to n}$ is the offloading ratio to AN-MEC neighbors, where $\forall n \in \mathcal{N}_i\backslash\{j\}$.
- $P^{C}_{i,j}$ is the offloading ratio from source AN-MEC to CN-MEC.
- $P^{CC}_{i\to m,j}$ is the offloading ratio to CN-MEC neighbors, where $\forall m \in \mathcal{M}\backslash\{i\}$.
- $P^{D}_{i,j}$ is the Offloading ratio to the cloud.

The action set, which is an output of the actor-network, is formulated as

$$a_t = \left\{P^{A}_{i,j}, P^{AA}_{i,j\to n}, \forall n \in \mathcal{N}_i\backslash\{j\}, P^{C}_{i,j}, P^{CC}_{i\to m,j}, \right.$$
$$\left. \forall m \in \mathcal{M}\backslash\{i\}, P^{D}_{i,j}\right\}^{M,N_i}_{i=1,j=1}. \tag{14}$$

Equation (14) applies to single-agent offloading. In multi-agent TD3, ith agent is responsible only for the offloading decision of traffic arriving at the jth AN-MEC. Thus, the action set for the ith agent is

$$a_t = \left\{P^{A}_{i,j}, P^{AA}_{i,j\to n}, \forall n \in \mathcal{N}_i\backslash\{j\}, P^{C}_{i,j}, P^{CC}_{i\to m,j}, \right.$$
$$\left. \forall m \in \mathcal{M}\backslash\{i\}, P^{D}_{i,j}\right\}^{N_i}_{i=1,j=x}. \tag{15}$$

3) Reward: In RL, the agent is not trained using labeled data sets but rather by providing a reward signal to indicate the quality of the action executed. Positive reward is given for actions that contribute to the system's objective. Otherwise, the agent's

Algorithm 1: Proposed MADRL-Based TD3 Algorithm.

```
1  Input: Number of Agents $N$, Arrival traffic $(\lambda_{i,j})$
2  Output: Offloading ratio
     $(P^{A}_{i,j}, P^{AA}_{i,j\rightarrow n}, P^{C}_{i,j}, P^{CC}_{i\rightarrow m,j}, P^{D}_{i,j})$
3  Initialize critic model $Q_i \dots Q_N$ where
     $(Q_i \leftarrow (Q_{\theta_{i,1}}, Q_{\theta_{i,2}}))$ and critic target model,
     $Q'_i \dots Q'_i$ where $(Q'_i \leftarrow Q_{\theta'_{i,1}}, Q_{\theta'_{i,2}})$ with random
     parameters $\theta_{i,1}, \theta_{i,2}$, and $\theta'_{i,2}, \theta'_{i,2}$;
4  Initialize actor model $\pi_{\phi_1} \dots \pi_{\phi_N}$ and actor target
     model $\pi'_{\phi_1} \dots \pi'_{\phi_N}$ with random parameters
     $\phi_1 \dots \phi_N$, and $\phi'_1 \dots \phi'_N$;
5  Initialize replay buffer $\mathcal{B}$;
6  for i=1 to N do
7      for t=1 to T do
8          select random action $a_{i,t} \sim \pi_{\phi_i}(s) + \epsilon$,
             $\epsilon \sim \mathcal{N}(0, \sigma)$ and observe reward $r_t$ and new
             state $s'_t$ ;
9          Store transition tuple $(s_t, a_t, s'_t, r_t)$ in $\mathcal{B}$ ;
10         Sample mini-batch of $\mathcal{N}$ transition
             $(s_t, a_t, s'_t, r_t)$ from $\mathcal{B}$ ;
11         $\tilde{a}_t \leftarrow \pi_{\phi'_i}(s'_t) + \epsilon$, $\epsilon \sim clip(\mathcal{N}(0, \tilde{\phi}_i), -c, c)$ ;
12         $y \leftarrow r_t + \gamma \left( \min_{i=1,2} Q_{\theta'_i}(s'_t, \tilde{a}_t) \right)$ ;
13         Update critics
             $\theta_i \leftarrow \mathrm{argmin}_{\theta_i} N^{-1} \sum \left(y - Q_{\theta_i}(s_t, a_t)\right)^2$
14         if t mod d then
15             Update actor parameter $\phi_i$ policy gradient:
               $\nabla_{\phi_i} J(\phi_i) =$
               $N^{-1} \sum \nabla_{a_t} Q_{\theta_{i,1}}(\mathbf{s_t}, a_t)\big|_{a_t=\pi_{\theta_i}(s_t)} \nabla_{\theta_i} \pi_{\theta_i}(s_t)$;
               Update target network:
16             $\theta'_i \leftarrow \tau\theta_i + (1-\tau)\theta'_i$
17             $\phi'_i \leftarrow \tau\phi_i + (1-\tau)\phi'_i$
18         end
19     end
20 end
```

reward will be negative. The goal of RL is to maximize the reward. Since the goal of offloading optimization is to reduce average latency, the reward for action in tth is calculated as

$$r_t = (Z_l(t)), \tag{16}$$

where $Z_l(t) = \frac{1}{l}$ and l is given in (8).

B. Simulated Annealing

Simulated annealing (SA) is a global optimization technique inspired by the annealing process in metallurgy, which avoids local optima by probabilistically accepting less desirable solutions in a controlled manner [27]. The SA-based offloading agent initially determines an offloading ratio (Y) for each AN-MEC site with incoming tasks. This ratio represents the distribution of tasks offloaded to AN-MEC sites, stored in an array of floats that sum to 1. SA uses temperature (T) to control the optimization process. At the start, SA has a high temperature, indicating a high probability of accepting suboptimal solutions. This likelihood decreases with each iteration as the temperature is gradually reduced. During each iteration, SA generates a new solution (Y_{new}), and the performance of this new solution $f(Y_{new})$ is compared to the previous one $f(Y)$. If the new solution performs better, it is retained. If not, the new solution may still be selected with a probability given by $\frac{1}{exp(\frac{f(Y_{new})-f(Y)}{T_t})}$, where T_t is the temperature at iteration t. The measured performance is the average latency experienced by incoming traffic.

C. Proposed MADRL-Based TD3 Algorithm

We investigated ratio-based offloading in a continuous action space under dynamic conditions. The TD3 algorithm, a state-of-the-art off-policy RL algorithm, is ideal for this due to its ability to mitigate overestimation and handle continuous action spaces [28]. However, as the number of sites increases, the input size grows, extending convergence time. To address this, we proposed a Multi-Agent Deep Reinforcement Learning (MADRL) approach based on TD3 for ratio-based offloading and faster decisions. By employing multiple agents, this method reduces input sizes for each agent, enabling faster convergence and more efficient processing. Multi-agent systems can collaborate and handle localized inputs, enhancing overall performance and adaptability. The MADRL approach optimizes the offloading ratio in a continuous action space, effectively navigating dynamic conditions and improving system latency and performance.

The proposed MADRL-based algorithm learns a Q-function and a policy. Each offloading agent in a cloud-edge environment locally observes and calculates a local offloading ratio, facilitated by the policy network. In this scenario, the service provider equips each central office with a CN-MEC site, connecting several AN-MEC base stations and the cloud. Equation (15) provides the offloading ratio in a multi-agent environment, highlighting the effectiveness of our approach. The Q-function (value function) assesses the quality of a state-action pair (s_t, a_t), while a policy takes the environmental state as input and returns an action (a_t). Algorithm 1 shows the outlines of the utilization of a pair of critic networks, a delayed update of the actor model, and the management of actor noise for the modified MADRL-based TD3 algorithm.

Algorithm 1 works as follows. Line 2 initializes two critic models with parameters $\theta_{i,1}, \theta_{i,2}$ and two critic targets with parameters $\theta'_{i,1}, \theta'_{i,2}$, where $i = 1, 2, \ldots, N$ denotes the number of agents. The actor model with parameter ϕ_i and the actor target model with parameter ϕ'_i are initialized in line 4. The actor model represents the entity that directly interacts with its environment by determining a_t in response to the received s_t. Both the critic and actor models are implemented as neural networks (NN). NNs are trained using a batch of $\mathcal{N}$ transitions stored in replay buffer $\mathcal{B}$, as shown in lines 9 and 10. As indicated in line 11, the actor target model predicts the next action $(\tilde{a}_t)$ by utilizing s'_t and adds exploratory noise to it. In line 12, the critic target model uses $(s'_t, \tilde{a}_t)$ to anticipate the future reward of (s_t, a_t). Target rewards are extracted from the critic target with the lowest rewards, multiplied by the discount factor γ, and added to the present reward. While line 13 is used to update the critic model by evaluating the pair (s_t, a_t). The state (s_t) represents the

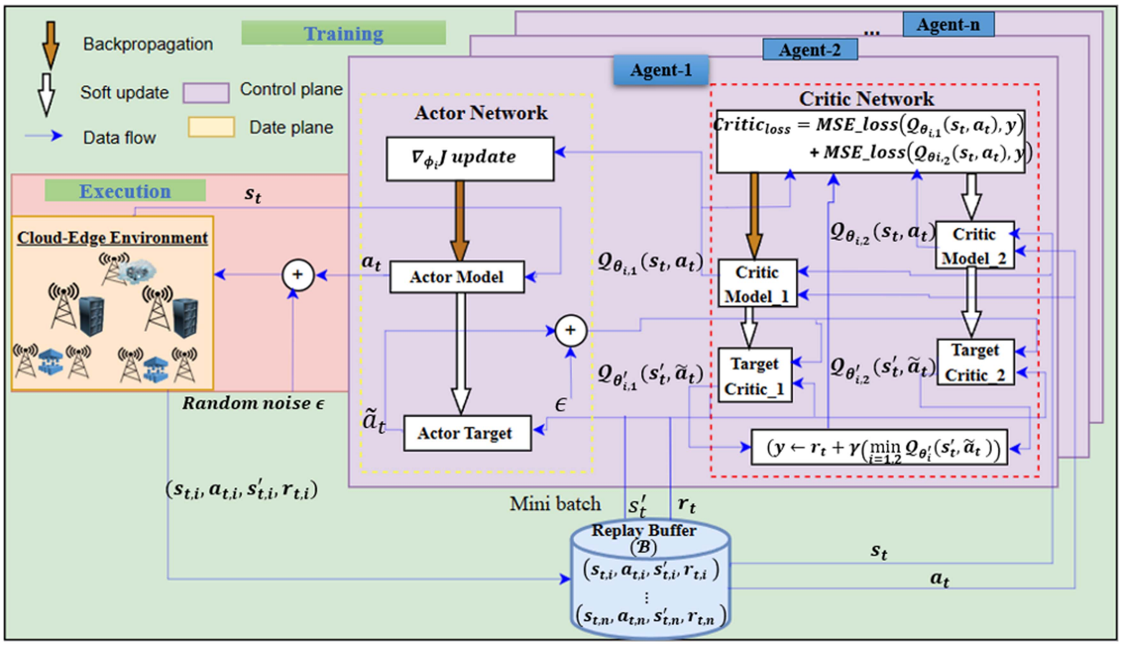

Backpropagation
Training
Soft update
Control plane
Data flow
Date plane
Agent-1
Agent-2
Agent-n
Actor Network
Critic Network
$\nabla_{\phi_i} J$ update
$Critic_{loss} = MSE\_loss(Q_{\theta_{i,1}}(s_t, a_t), y) + MSE\_loss(Q_{\theta_{i,2}}(s_t, a_t), y)$
Execution
s_t
Cloud-Edge Environment
a_t
Actor Model
$Q_{\theta_{i,1}}(s_t, a_t)$
$Q_{\theta_{i,2}}(s_t, a_t)$
Critic Model_1
Critic Model_2
Target Critic_1
Target Critic_2
$Q_{\theta'_{i,1}}(s'_t, \tilde{a}_t)$
$Q_{\theta'_{i,2}}(s'_t, \tilde{a}_t)$
Random noise ϵ
ϵ
$\tilde{a}_t$
Actor Target
$(y \leftarrow r_t + \gamma(\min_{i=1,2} Q_{\theta'_i}(s'_t, \tilde{a}_t)))$
$(s_{t,i}, a_{t,i}, s'_{t,i}, r_{t,i})$
Mini batch
s'_t
r_t
Replay Buffer ($\mathcal{B}$)
$(s_{t,i}, a_{t,i}, s'_{t,i}, r_{t,i})$
$(s_{t,n}, a_{t,n}, s'_{t,n}, r_{t,n})$
s_t
a_t

Fig. 3. MADRL-based TD3 offloading structure.

necessary cloud-edge environment information for action determination, and action (a_t) is the offloading ratio used to distribute incoming traffic to cloud-edge sites. The minimum mean square error (MSE) between the critic model's reward and the target reward is subsequently calculated. This error serves as the basis for updating the twin-critic models through back-propagation delay. Lines 14–18 illustrate a phase of steady learning, during which the update of the actor model and all target networks is postponed. In line 15, the actor model is refined using gradient descent with respect to the value estimates provided by a critic model. Lines 16–18 indicate all target networks are refreshed by duplicating the model parameters with a factor of τ to slow down the network update process and maintain a minimal MSE.

Fig. 3 depicts the MADRL-based TD3 structure, consisting of three key components: the cloud-edge environment, the replay buffer, and the agents. The execution and training processes occur simultaneously. During execution, the actor model interacts directly with the cloud-edge environment, which provides the agent with essential information for determining the offloading ratio at time t iterations. This information includes node latency, link latency, communication capacity, and computing capacity, referred to as states s_t as in (13).

The replay buffer is used to store the agent-environment interaction history as a tuple of state, action, next state, and reward (s_t, a_t, s'_t, r_t). These data update the agent model's parameters throughout the training phase. The models are trained using batches of acquired data instead of waiting for a large accumulation of data in the replay buffer $(\mathcal{B})$.

The cloud-edge offloading agents (Agent-1, Agent-2, ..., Agent-n) determine the optimal offloading option to minimize system latency l. The aim is to maximize the cumulative system reward as given in (16). Each cloud-edge offloading agent uses six neural networks: two actor networks (Actor model and Actor target) and four critic networks (twin critic models and twin critic targets). The actor model is responsible for determining the optimal offloading ratio, which includes both the destination of the offloaded traffic (vertical and horizontal directions) and the amount of traffic that would be offloaded. The critic model is used to evaluate the pair (s_t, a_t), while the critic target uses $(s'_t, \tilde{a}_t)$ to calculate the future reward of (s_t, a_t). Updating policies and Q-value networks for all offloading agents occurs through a training procedure that utilizes global cloud-edge system data stored in a replay buffer $(\mathcal{B})$.

V. Simulation and Results

A simulation has been created to evaluate the comparison between conventional optimization approaches and RL-based offloading algorithms in the given cloud-edge federated systems. The existing network control plane utilizes a network abstraction [28] similar to simulation models, along with optimization techniques based on these abstractions.

A. Parameter Settings

Tables III and IV detail the system and agent parameters, respectively. We developed the DRL environment simulation using Python. These experiments are available on GitHub.[1] All AN-MECs were randomly distributed based on base stations, typically placed according to population density using Epitools [29]. We considered up to five CN-MECs, with each CN-MEC mapped to eight AN-MEC sites. As noted in [8], the distance between two AN-MECs ranges from 30 to 60 km, while the distance between two CN-MECs ranges from 40 to 150 km.

[1]https://github.com/seifuGthab/Cloud-edge-simulation-codes.

TABLE III
PARAMETER SETTINGS

Parameters	Notations	Values
Topology:		
Number of AN-MEC sites	$\mathcal{N}_i$	8
Number of CN-MEC sites	$\mathcal{M}$	1, 2, 3, 4, 5
Cloud	D	1
Total number of sites	$M(N_i) + M + D$	10, 19, 28, 37, 46
Distance between AN-MECs	$d^{AA}_{i,j\leftrightarrow n}$	≈ 30 km – 60 km
Distance between AN-MEC and CN-MEC	$d^{AC}_{i,j}$	≈ 30 km – 60 km
Distance between CN-MECs	$d^{CC}_{i\leftrightarrow m}$	≈ 40 km – 150 km
Distance between CN-MEC and D	d^{CD}_{i}	≈ 100 km – 1000 km
Capacity:		
AN-MEC site	$\mu^{A}_{i,j}$	3×10^4packets/s
CN-MEC site	μ^{C}_{i}	2.7×10^5packets/s
Cloud	μ^{D}	3×10^6packets/s
Uplink, Downlink	$B^{UA}_{i,j}, B^{AU}_{i,j}$	1 Gbps
Traffic:		
Normal traffic rate	$\lambda_{i,j}$	10 K, 20 K, 30 K, 40 K packet/s
Hotspot traffic rate	$\lambda_{i,j}$	20 K, 40 K, 60 K, 80 K packet/s
Packet size	p	10 Kb

TABLE IV
SA AND TD3 ALGORITHM PARAMETER SETTINGS

Parameters	Value
SA algorithm Parameter settings	
Initial temperature (T)	1000
Number to decrease temperatures (M)	300
Neighbors to visit at a temp. (N)	30
Cooling parameter (c)	0.99
Search step (α)	0.1
TD3 algorithm Parameter settings	
Optimizer	ADAM
Activation function	Softmax
Batch size	100
Exploration noise	0.1
Discount factor (γ)	0.99
Learning rate (τ)	0.005
Policy noise (ϵ)	0.05
Noise clip	0.5
Policy network update	2
Evaluation frequency	100

The distance between an AN-MEC and its corresponding CN-MEC ranges from 30 to 60 km, whereas the distance between a CN-MEC and the cloud ranges from 100 to 1000 km.

The traffic arrival rate followed a Poisson process, and service times were exponentially distributed, reflecting real network traffic patterns. We considered two traffic arrival scenarios: normal traffic with loads of 10 K, 20 K, 30 K, and 40 K packets per second (packets/s), and hotspot traffic with double these rates to simulate peak conditions [8]. A fixed packet size (p) of 10 Kb was used throughout the experiments to ensure consistency in data handling and performance evaluation. In our architecture, CN-MEC servers are positioned at the Internet Service Provider (ISP) [30] and supported by central office (CO) infrastructure, which facilitates robust network device deployment [31]. These servers have a capacity of 2.7×10^5 packets/s, which is higher than the AN-MEC servers. AN-MEC servers, constrained by their location behind base stations with limited space and cooling capabilities, handle only 3×10^4 packets/s. The cloud server has a capacity of 3×10^6 packets/s, to handle high volumes of data traffic.

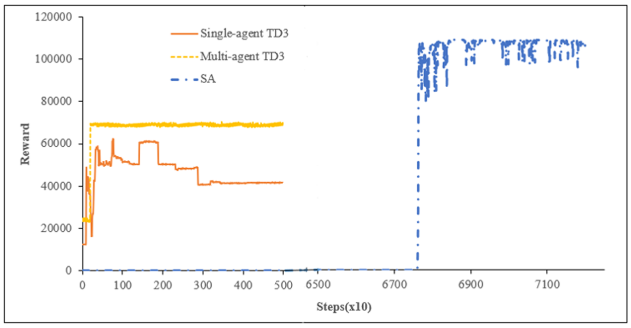


Fig. 4. Algorithm's learning curve.

Table IV presents the specific configurations for the offloading agents for SA and DRL. To address this, we used Optuna [32], an automatic hyperparameter optimization framework, to determine the optimal hyperparameter values for our simulation. The Softmax activation function was applied to the actor-network, forming a categorical distribution that assigns inputs to categories. For both the single-agent and multi-agent actor-networks, the state (as defined in (13)) was used as input to compute an action, for all MEC sites (as calculated in (14) and (15), respectively). We used discount factor (γ) of 0.99 and learning rate (τ) of 0.005 in our experiments. In conclusion, our framework optimizes performance across a multi-tier edge and cloud infrastructure by considering specific arrival rates and server capacities reflective of real-world scenarios. This approach ensures a robust simulation of different task data volumes and computational loads.

B. Results Analysis

The experimental results are organized to address the specified research issues, such as decision time, convergence time, decision quality (or service performance), offloading under unpredicted parameters in the context of RL vs. SA, and a comparison between single-agent and multi-agent offloading. Since the SA algorithm is based on a probabilistic technique, we used it as a benchmark to compare our approach. However, brute force algorithms are impractical to use in this kind of problem, due to the continuous action space.

1) Evaluation of Learning Curves: Fig. 4 illustrates the evaluation of learning curves concerning the algorithm's ability to adapt quickly to new traffic patterns. We conducted 50 000 episodes (5000 steps) for multi-agent TD3 and single-agent TD3 to assess this learning curve. For SA, approximately 7 10 000 iterations (71 000 steps) were employed. In RL, a step represents a complete interaction cycle between the agent and its environment, including the agent's action, the environment's response, and the agent's learning from the outcome. In SA, a step refers to a single iteration or move within the optimization search space. The multi-agent TD3 and single-agent TD3 required 500–1000 steps to approximate the maximum reward, while SA required 67 500 steps. This difference arises because of the sequential optimization of AN-MECs' offloading decisions by

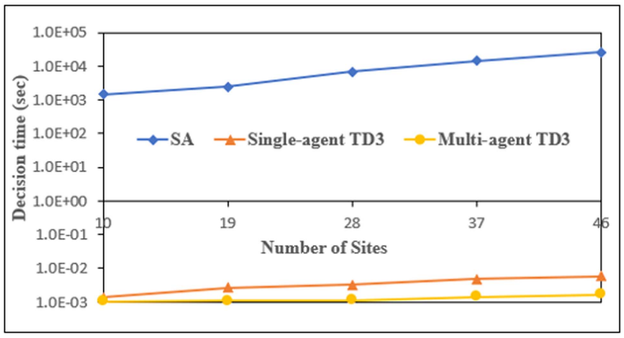
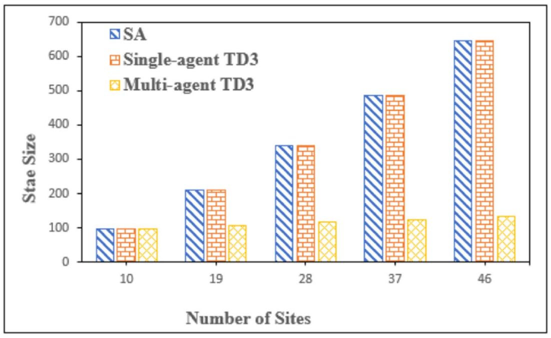


Fig. 5. Decision time.

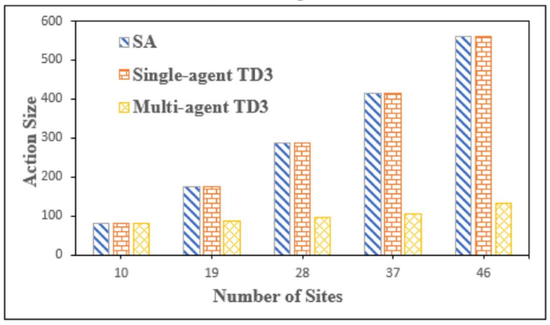
SA, including unvisited sites with arrival traffic that exceeds their capacity due to the initial random offloading mechanism, resulting in significant delay and nearly zero reward. The multi-agent TD3 demonstrated a faster learning curve compared to the single-agent TD3 approach because it focused only on the information required by each agent. This necessitated the removal of unnecessary features from the agent model training, such as information about AN-MEC sites connected to other CN-MECs and the cloud that could not serve as offloading destinations. Additionally, it had smaller input and output dimensions and yielded more stable results after 500 steps compared to single-agent TD3.

2) Decision Time: Decision time refers to the duration required to determine an offloading action, also known as response time. Since the control plane makes the offloading decision, fast decision-making becomes imperative. Fig. 5 compares the decision times for SA, single-agent TD3, and multi-agent TD3 offloading. Before selecting an action, the SA employing exhaustive search must first converge. The optimal search for a decision by SA takes thousands of seconds. With the increase in sites from 10 to 46, the decision time increases exponentially from 1447 to 26 177 seconds thereby expanding the search space for alternative solutions. In single-agent and multi-agent TD3 offloading, the execution and training phases occur independently, with the decision to offload made before convergence. TD3 could make decisions in milliseconds, which is six orders of magnitude faster than the SA algorithm. The decision time depends on the input and output sizes of the actor NN, which maps system information into actions.

Fig. 6 illustrates the state (input) and action (output) sizes, supporting our analysis of the decision times for the SA, single-agent TD3, and multi-agent TD3 algorithms. Fig. 6(a) and (b) show that the state and action sizes for the SA and single-agent TD3 are larger than those for the multi-agent TD3. This is because the multi-agent TD3 only needs to consider local system information to determine the optimal offloading ratio for all AN-MEC sites. While the input state and action spaces of SA and single-agent TD3 increase linearly with the growing number of sites, those of multi-agent TD3 show marginal growth as they deal exclusively with local network information. These input and output sizes impact the number of neurons employed by the neural networks. To decide on an offloading ratio, the



(a) State space.



(b) Action space.

Fig. 6. State and Action space.

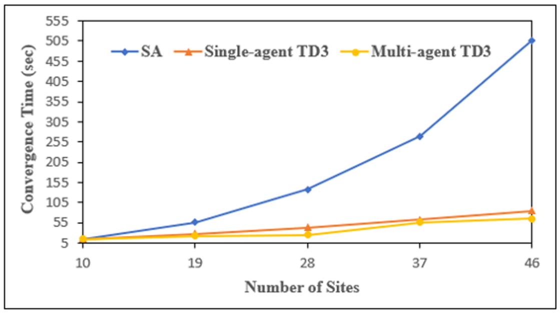


Fig. 7. Convergence time.

single-agent and multi-agent TD3 take 5.5 to 1.5 milliseconds and 1.68 to 1.06 milliseconds, respectively, as the expansion of sites from 10 to 46. In a multi-agent environment, the number of agents expands by the number of AN-MEC sites, with the number of neurons moderately affected by additional CN-MEC data processed by an offloading agent in an AN-MEC site.

DRL-based techniques are preferred in control plane decisions because they can make decisions in milliseconds or seconds, accommodating fluctuating traffic. DRL models require pre-training as they rely on trial and error, which may take considerable time to converge on optimal decisions. Despite SA choices taking tens of minutes, which might become outdated due to prevailing traffic volumes, decisions made within an hour are more likely to be inaccurate.

3) Convergence Time: Convergence time refers to the duration needed for an algorithm to generate optimal results. Fig. 7 compares the convergence times of SA, single-agent TD3,

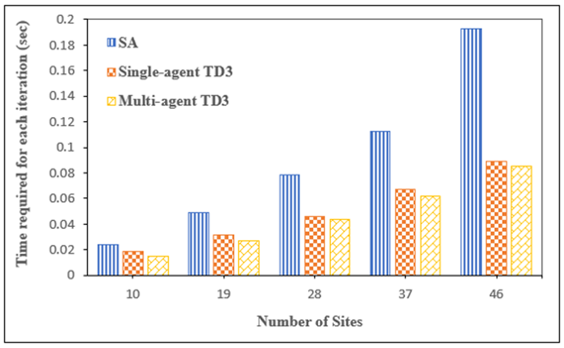


Fig. 8. Step time.

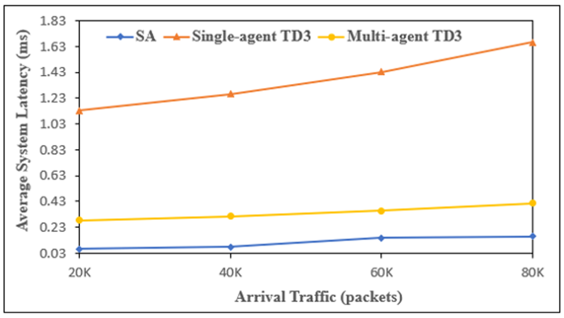


Fig. 9. Offloading performance.

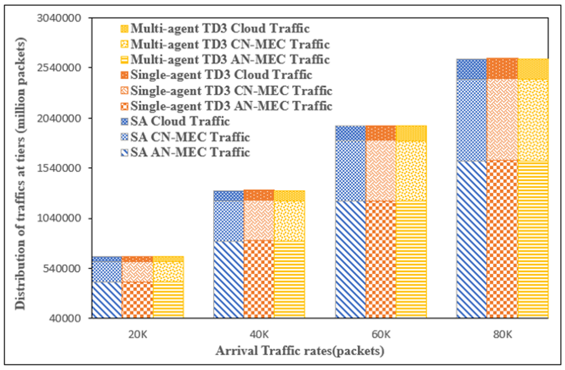


Fig. 10. Distribution of arrival traffic.

and multi-agent TD3 optimizations. The TD3 agent initially employed weighted offloading, as using a random float for the initial offloading ratio would take considerable time to converge. TD3 exhibited a shorter convergence time than SA because the model was pre-trained with weighted-based offloading. With 10 sites, the TD3 single-agent and multi-agent required 15 and 14 seconds to converge, respectively. With 46 sites, these times increased to 86 and 64 seconds, respectively. In contrast, SA needed 15 seconds to converge with 10 sites and a significant 507 seconds with 46 sites.

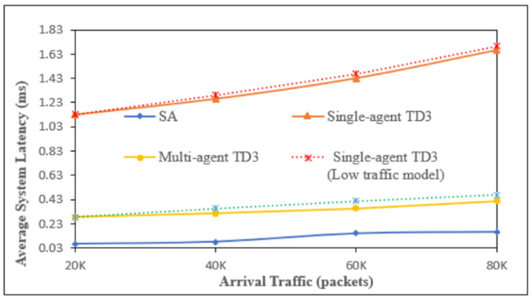
Fig. 8 shows the time for each step. The SA step requires more time compared to the TD3 step during training. This is because the SA step calculates both the current and new solution performance and evaluates the chance of selecting a sub-optimal solution. Since single-agent and multi-agent TD3 models are trained using global data and share the same replay buffer, their step times during training are nearly identical.

The input size affects algorithm complexity, influenced by the number of servers and sites. For the SA algorithm, the time complexity is $O(n^{k \times l})$, where n is the number of sites. This complexity arises because each site's state may need to be evaluated or modified. The parameter k represents the cooling schedule, indicating the number of temperature reductions and iterations. The parameter l denotes the number of neighboring solutions evaluated at each temperature level. Although SA doesn't exhaustively search the solution space, it evaluates many neighbors per temperature step, thus incorporating l into the complexity.

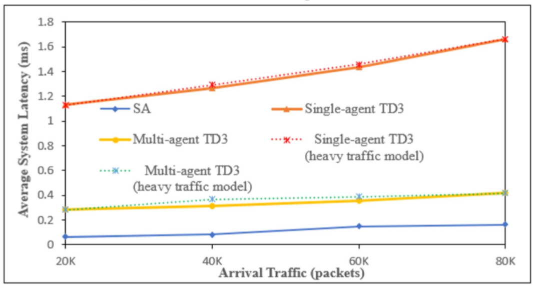
For DRL approaches, the time complexity depends on the input size (n), which is the number of sites, the total number of training steps (t). The parameter h represents the depth of the neural network used for offloading. Each neural network layer involves a matrix multiplication with complexity $O(m^3)$, where m is the number of neurons per layer. Consequently, the overall complexity of training the neural network can be expressed as $O(n \times t \times h \times m^3)$. The training procedure for the neural network shows a linear relationship with the input size n.

4) Decision Quality: Fig. 9 compares the performance of offloading agents based on SA, single-agent, and multi-agent TD3. The SA that performs exhaustive searching outperforms all others in all traffic scenarios. The single-agent and multi-agent TD3 had latency of only 1.5 and 0.2 milliseconds greater than the SA, respectively. Due to their low propagation delay, locally available resources are more likely to be offloading destinations than distant ones. multi-agent TD3, which determines the offloading ratio of a CN-MEC, had a smaller search space for actions and outperformed single-agent TD3, which had a larger search space for actions. For action determination, the actor model of single-agent TD3 also considered global information. In multi-agent TD3, an offloading agent may not require certain information, such as knowledge about a remote AN-MEC site.

Fig. 10 shows the traffic distribution at different arrival rates for various methods. The bars are stacked to show the cumulative distribution of other types of traffic (e.g., Multi-agent TD3 Cloud Traffic, Single-agent TD3 Cloud Traffic, SA Cloud Traffic, etc.) at different arrival rates. Each segment of a bar represents a specific traffic type, and their combined height indicates the total traffic. Most traffic for SA and TD3-based agents was offloaded to AN-MEC locations. As the traffic rate increased from 40 K to 80 K, traffic offloaded to CN-MEC sites and the Cloud also increased, since CN-MEC sites and the cloud were more centrally located and had greater capacity than AN-MEC facilities. In a queuing system, a larger centralized capacity performs better in terms of computing delay than a scattered capacity with a smaller capacity [8]. In this experiment, the ratio of traffic offloaded to AN-MEC, CN-MEC, and cloud sites was nearly the same for all traffic rates with the ratios being 61.53%, 30.82%, and 7.65%, for AN-MEC, CN-MEC, and cloud, respectively. Despite SA-based offloading having the lowest average latency, it takes minutes or even hours to make a decision, which is insufficient to accommodate the fluctuating traffic rate. A TD3-based algorithm with similar orders of magnitude for average latency as SA could determine offloading ratios five



(a) Low traffic model performance.



(b) Heavy traffic model performance.

Fig. 11. Effect of fluctuating traffic to the agent model.

orders of magnitude faster and accommodate fluctuating traffic rates in seconds.

5) Decision With Unpredicted Parameter: The agent may encounter unpredicted parameters, for instance, consider a scenario where an agent model trained on light traffic suddenly faces fluctuating heavy traffic. If the environmental conditions change, SA methods will recalculate the offloading decision. Fig. 11 illustrate the responsiveness to traffic fluctuations. Interestingly, the model trained in heavy traffic demonstrated more efficient handling of light traffic compared to the model trained in light traffic when facing heavy traffic conditions. Fig. 11(a) shows that the single-agent TD3 model trained for light traffic performs slightly worse with high arrival traffic. In contrast, Fig. 11(b) highlights that models trained in heavy traffic provide more accurate offloading ratios and lower average latency compared to those trained for low traffic.

VI. Conclusion

We presented a ratio-based offloading optimization approach for cloud-edge systems, in which the control plane makes the decisions. The ratio-based offloading optimization for the cloud-edge system includes a continuous action search space for determining the total amount of traffic that needs to be offloaded. Traditional optimization methods, like SA, require 1,477 to 26 177 seconds to make decisions as the number of sites increases. However, these decision times violate the requirements of the control plane, which needs to determine the offloading ratio within less than seconds in response to fluctuating arrival traffic. For both single-agent and multi-agent TD3 takes 1.5 to 5.5 ms and 1.06 to 1.68 ms, respectively, to decide on an offloading ratio. Notably, the decision time of the TD3-based approach is six orders of magnitude faster than that of the SA approach. While single-agent and multi-agent TD3 introduce average latency of 1.5 and 0.2 ms longer than SA, respectively, they manage to approximate the decision quality of SA. Compared to multi-agent with single-agent, multi-agent can provide more accurate and faster decision and convergence time than single-agent. The model trained with heavy arrival traffic rates offers greater adaptability to various arrival rate patterns. If the ratio of heavy traffic rates remains within the limits of edge and cloud capacity, the model produces similar results when applied to lower arrival rates.

In future work, it is crucial to consider the resource requirements for allocating resources to MEC sites to minimize long waiting delays from insufficient processing node resources. Deploying the proposed approach in practical scenarios will provide valuable insights. We plan to refine our model by incorporating detailed server characteristics and dynamic offloading strategies to better represent real-world scenarios. Another potential extension of our work could be the integration of explainable AI techniques to enhance the transparency and interpretability of our MADRL-based solutions.

References

[1] T. Mengistu, A. Alahmadi, A. Albuali, Y. Alsenani, and D. Che, "A "no data center" solution to cloud computing," in *Proc. IEEE 10th Int. Conf. Cloud Comput.*, 2017, pp. 714–717.

[2] S. Long, Y. Zhang, Q. Deng, T. Pei, J. Ouyang, and Z. Xia, "An efficient task offloading approach based on multi-objective evolutionary algorithm in cloud-edge collaborative environment," *IEEE Trans. Netw. Sci. Eng.*, vol. 10, no. 2, pp. 645–657, Mar./Apr. 2023.

[3] B. Kar, Y.-D. Lin, and Y.-C. Lai, "Cost optimization of omnidirectional offloading in two-tier cloud–edge federated systems," *J. Netw. Comput. Appl.*, vol. 215, 2023, Art. no. 103630.

[4] T. Wang et al., "An intelligent dynamic offloading from cloud to edge for smart IoT systems with Big Data," *IEEE Trans. Netw. Sci. Eng.*, vol. 7, no. 4, pp. 2598–2607, Fourth Quarter 2020.

[5] G. Wu, H. Wang, H. Zhang, Y. Zhao, S. Yu, and S. Shen, "Computation offloading method using stochastic games for software defined network-based multi-agent mobile edge computing," *IEEE Internet Things J.*, vol. 10, no. 20, pp. 17620–17634, Oct. 2023.

[6] G. Wu et al., "Combining Lyapunov optimization with actor-critic networks for privacy-aware IIoT computation offloading," *IEEE Internet Things J.*, vol. 11, no. 10, pp. 17437–17452, May 2024.

[7] B. Kar, Y.-D. Lin, and Y.-C. Lai, "OMNI: Omni-directional dual cost optimization of two-tier federated cloud-edge systems," in *Proc. Int. Conf. Commun. (ICC)*, IEEE, 2020, pp. 1–7.

[8] W. Yahya, E. Oki, Y.-D. Lin, and Y.-C. Lai, "Scaling and offloading optimization in Pre-CORD and Post-CORD multi-access edge computing," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 4, pp. 4503–4516, Dec. 2021.

[9] L. Peterson et al., "Central office re-architected as a data center," *IEEE Commun. Mag.*, vol. 54, no. 10, pp. 96–101, Oct. 2016.

[10] A. Suzuki, M. Kobayashi, and E. Oki, "Multi-agent deep reinforcement learning for cooperative computing offloading and route optimization in multi cloud-edge networks," *IEEE Trans. Netw. Service Manag.*, vol. 20, no. 4, pp. 4416–4434, Dec. 2023.

[11] G. Wu, X. Chen, Z. Gao, H. Zhang, S. Yu, and S. Shen, "Privacy-preserving offloading scheme in multi-access mobile edge computing based on MADRL," *J. Parallel Distrib. Comput.*, vol. 183, 2024, Art. no. 104775.

[12] Y.-D. Lin, W. Yahya, C.-T. Wang, C.-Y. Li, and J. H. Tseng, "Scalable mobile edge computing: A two-tier multi-site multi-server architecture with autoscaling and offloading," *IEEE Wireless Commun.*, vol. 28, no. 6, pp. 168–175, Dec. 2021.

[13] B. Kar, W. Yahya, Y.-D. Lin, and A. Ali, "Offloading using traditional optimization and machine learning in federated cloud-edge-fog systems: A survey," *IEEE Commun. Surveys Tut.*, vol. 25, no. 2, pp. 1199–1226, Second Quarter 2023.

[14] M. Khalil and S. A. Khan, "Control plane and data plane issues in network layer multipathing," in *Proc. 2020 Int. Conf. Inf. Technol. Syst. Innov.*, 2020, pp. 263–269.

[15] A. Shakarami, M. Ghobaei-Arani, and A. Shahidinejad, “A survey on the computation offloading approaches in mobile edge computing: A machine learning-based perspective,” *Comput. Netw.*, vol. 182, 2020, Art. no. 107496.
[16] H. Lee and J. Jeong, “Multi-agent deep reinforcement learning (MADRL) meets multi-user MIMO systems,” in *Proc. 2021 IEEE Glob. Commun. Conf.*, 2021, pp. 1–6.
[17] J. Liu, X. Wang, S. Shen, G. Yue, S. Yu, and M. Li, “A Bayesian Q-learning game for dependable task offloading against DDoS attacks in sensor edge cloud,” *IEEE Internet Things J.*, vol. 8, no. 9, pp. 7546–7561, May 2021.
[18] X. Qiu, L. Liu, W. Chen, Z. Hong, and Z. Zheng, “Online deep reinforcement learning for computation offloading in blockchain-empowered mobile edge computing,” *IEEE Trans. Veh. Technol.*, vol. 68, no. 8, pp. 8050–8062, Aug. 2019.
[19] M. Khayyat, I. A. Elgendy, A. Muthanna, A. S. Alshahrani, S. Alharbi, and A. Koucheryavy, “Advanced deep learning-based computational offloading for multilevel vehicular edge-cloud computing networks,” *IEEE Access*, vol. 8, pp. 137052–137062, 2020.
[20] G. Qu, H. Wu, R. Li, and P. Jiao, “DMRO: A deep meta reinforcement learning-based task offloading framework for edge-cloud computing,” *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 3, pp. 3448–3459, Sep. 2021.
[21] J. Wang, J. Hu, G. Min, A. Y. Zomaya, and N. Georgalas, “Fast adaptive task offloading in edge computing based on meta reinforcement learning,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 1, pp. 242–253, Jan. 2021.
[22] H. Lu, C. Gu, F. Luo, W. Ding, and X. Liu, “Optimization of lightweight task offloading strategy for mobile edge computing based on deep reinforcement learning,” *Future Gener. Comput. Syst.*, vol. 102, pp. 847–861, 2020.
[23] Z. Ali, L. Jiao, T. Baker, G. Abbas, Z. H. Abbas, and S. Khaf, “A deep learning approach for energy efficient computational offloading in mobile edge computing,” *IEEE Access*, vol. 7, pp. 149623–149633, 2019.
[24] G. Wu, Z. Xu, H. Zhang, S. Shen, and S. Yu, “Multi-agent DRL for joint completion delay and energy consumption with queuing theory in MEC-based IIoT,” *J. Parallel Distrib. Comput.*, vol. 176, pp. 80–94, 2023.
[25] L. Ale, N. Zhang, X. Fang, X. Chen, S. Wu, and L. Li, “Delay-aware and energy-efficient computation offloading in mobile-edge computing using deep reinforcement learning,” *IEEE Trans. Cogn. Commun. Netw.*, vol. 7, no. 3, pp. 881–892, Sep. 2021.
[26] S. Fujimoto, H. Hoof, and D. Meger, “Addressing function approximation error in actor-critic methods,” in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 1587–1596.
[27] L. R. Rere, M. I. Fanany, and A. M. Arymurthy, “Simulated annealing algorithm for deep learning,” *Procedia Comput. Sci.*, vol. 72, pp. 137–144, 2015.
[28] D. Raghunathan, R. Beckett, A. Gupta, and D. Walker, “ACORN: Network control plane abstraction using route nondeterminism,” in *Proc. Conf. Formal Methods Comput.-Aided Des.*, 2022, pp. 261–272.
[29] E. Sergeant, “Epitools epidemiological calculators,” *Australia: Ausvet*. 2018. Accessed: Nov. 11, 2024. [Online]. Available: https://epitools.ausvet.com.au/
[30] S. Secci, P. Raad, and P. Gallard, “Linking virtual machine mobility to user mobility,” *IEEE Trans. Netw. Service Manag.*, vol. 13, no. 4, pp. 927–940, Dec. 2016.
[31] J. Jia, F. Xu, and Y. Yang, “Edge servers deployment based on 5G micro base stations,” in *Proc. 2021 Int. Conf. Comput. Netw. Electron. Autom.*, 2021, pp. 148–152.
[32] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, “Optuna: A next-generation hyperparameter optimization framework,” in *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2019, pp. 2623–2631.

Seifu Birhanu Tadele is currently working toward the Ph.D. degree with the National Taiwan University of Science and Technology (NTUST), Taipei, Taiwan. His research interests include cloud computing, multi-access edge computing (MEC), fog computing, optimization, and deep reinforcement learning.

Widhi Yahya received the Ph.D. degree from Electrical Engineering and Computer Science (EECS) International Graduate Program, National Yang Ming Chiao Tung (NYCU), Hsinchu City, Taiwan, in 2023. He is currently a Lecturer and Researcher of computer science with the University of Brawijaya, Malang, Indonesia. His research interests include network programming, software-defined networking, and multi-access edge computing (MEC) optimization.

Binayak Kar (Member, IEEE) received the Ph.D. degree in computer science and information engineering from the National Central University (NCU), Taoyuan City, Taiwan, in 2018. From 2018 to 2019, he was a Postdoctoral Research Fellow of computer science with the National Chiao Tung University (NCTU), Hsinchu City, Taiwan. He is currently an Assistant Professor of computer science and information engineering with the National Taiwan University of Science and Technology (NTUST), Taipei, Taiwan. His research interests include edge computing, cybersecurity, and quantum computing.

Ying-Dar Lin (Fellow, IEEE) received the Ph.D. degree in computer science from the University of California at Los Angeles (UCLA), Los Angeles, CA, USA, in 1993. He is currently a Chair Professor of computer science with the National Yang Ming Chiao Tung University (NYCU), Hsinchu, Taiwan. His work on multi-hop cellular was the first along this line and has been cited more than 1000 times. His research interests include network softwarization, cybersecurity, and wireless communications. He was or is on the Editorial Boards of several IEEE journals and magazines, and was the Editor-in-Chief of IEEE Communications Surveys and Tutorials (COMST), during 2016–2020. He is also an IEEE Distinguished Lecturer.

Yuan-Cheng Lai received the Ph.D. degree from the Department of Computer and Information Science, National Chiao Tung University, Hsinchu, Taiwan, in 1997. In August 2001, he joined the Faculty of the Department of Information Management with the National Taiwan University of Science and Technology and has been a Professor since February 2008. His research interests include performance analysis, wireless networks, network security, and ML.

Frezer Guteta Wakgra is currently working toward the Ph.D. degree with the National Taiwan University of Science and Technology (NTUST), Taipei, Taiwan. His research interests include cloud computing, multi-access edge computing (MEC), fog computing, optimization, and deep reinforcement learning.