Deep-Reinforcement-Learning-Based Resource Allocation for Cloud Gaming via Edge Computing

Xiaoheng Deng, *Member, IEEE*, Jingjing Zhang, Honggang Zhang, *Member, IEEE*, and Ping Jiang

***Abstract*—Compared with cloud computing, edge computing is capable of effectively solving the high latency problem in cloud gaming. However, there are still several challenges to address for optimizing system performance. On the one hand, the unpredictable bursts of game requests can cause server overload and network congestion. On the other hand, the mobility of players makes the system highly dynamic. Although existing research has studied game fairness and latency separately to improve the Quality of Experience (QoE), a tradeoff between fairness and latency has been largely ignored. Furthermore, how to balance network and computing load is identified as another constraint during optimization. Focusing on latency, fairness, and load balance simultaneously, we propose an adaptive resource allocation strategy through deep reinforcement learning (DRL) for a dynamic gaming system. The experimental results have demonstrated that the proposed algorithm outperforms the traditional optimization methods and classical reinforcement learning algorithms in solving complex multimodal reward problems.**

***Index Terms*—Cloud gaming, deep reinforcement learning (DRL), edge computing, software-defined networking.**

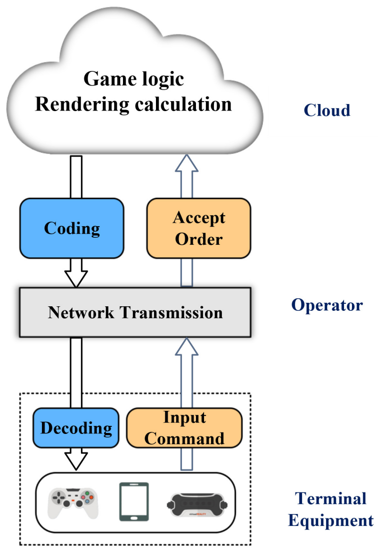


Fig. 1. Cloud gaming diagram.

I. Introduction

In recent years, with the rapid development of information technology and economy, computer games have become a popular way for people to entertain, compete, and socialize. However, traditional electronic games face challenges, such as limited hardware conditions, few usage scenarios, high cross-platform development costs, and slow growth in user scale [1]. As an effort to overcome these challenges, cloud games have attracted a considerable amount of attention. As shown in Fig. 1, during the run time of cloud games, complex game logic runs on high-performance cloud servers, which then transmit rendered pictures to game participants through the Internet. With the cloud gaming technology, game participants can use mobile devices with low-cost software to replace expensive and bulky hardware equipment. Cloud gaming reduces the cost of upgrading and capacity expansion of hardware equipment, and it is convenient to play cross-platform games whenever and wherever possible. Furthermore, cloud gaming relies on video stream, and game participants do not need to download various data packages from the Internet from a security perspective, so various game plug-ins can be avoided [1]. With such advantages, the use of cloud gaming services has increased significantly in recent years, and there is a large increase in cloud gaming all over the world. The global earning of cloud gaming is expected to reach 3.2 billion by 2023 [2]. In addition, the utilization of cloud gaming in new application scenarios, such as live broadcasting, advertising, and education unlocks the critical bottleneck of development.

However, the further development of cloud gaming is facing challenges, and it is worthwhile devoting much effort to addressing them.

A. High Bandwidth Consumption

High-quality game graphics consume lots of bandwidth. Cloud games transmit large amounts of game graphics to

Manuscript received 19 June 2022; revised 22 October 2022; accepted 10 November 2022. Date of publication 16 November 2022; date of current version 7 March 2023. This work was supported in part by the National Natural Science Foundation of China under Project 62172441, Project 62172449, and Project 61772553; in part by the Local Science and Technology Developing Foundation guided by Central Government (Free Exploration Project) under Grant 2021Szvup166; in part by the Opening Project of State Key Laboratory of Nickel and Cobalt Resources Comprehensive Utilization under Grant GZSYS-KY-2020-033; and in part by the Fundamental Research Funds for the Central Universities of Central South University under Grant 2021zzts0201. *(Corresponding author: Xiaoheng Deng.)*

Xiaoheng Deng, Jingjing Zhang, and Ping Jiang are with the School of Computer Science and Engineering and the Research Institute of Shenzhen, Central South University, Changsha 410083, China (e-mail: dxh@csu.edu.cn; z.jingjing@csu.edu.cn; pjiang@csu.edu.cn).

Honggang Zhang is with the Department of Engineering, University of Massachusetts Boston, Boston, MA 02125 USA (e-mail: honggang.zhang@umb.edu).

Digital Object Identifier 10.1109/JIOT.2022.3222210

game participants, which results in higher bandwidth requirements for cloud games than a regular video stream. Low network bandwidth can cause delays in in-game interactions and reduce the quality of cloud gaming graphics. For example, high-definition cloud games (1080p 30 frames/s) under H264 encoding conditions can reach code rates of around 8 Mb/s, requiring at least 10 Mb/s network bandwidth to ensure stable and smooth operation of the game [3]. With the expansion of application terminals to large-screen TVs and VRs, cloud games have gradually increased requirements for resolution and frame rate, and bandwidth consumption has also increased. Cloud games with 4K high-definition graphics need at least 80 Mb/s to run smoothly [1].

B. High Latency Response

The interaction delay in cloud gaming includes network delay and server response delay [4], of which network delay accounts for more than 70% of the total delay [3]. Traditional online gaming systems typically render actions on a player's local system before the player accesses the game server, thus hiding the impact of interaction latency. With cloud gaming, the rendering process is all implemented in the cloud platform, and the client cannot hide interaction latency by operating locally [1]. This feature of cloud games makes them more delay-sensitive than traditional games. For example, games with strict operational response latency requirements (action, combat, first-person shooting, and racing) typically need to keep latency under 80 ms. In addition, the significant latency of VR cloud games can cause motion sickness. Therefore, VR cloud games need to keep latency to 20 ms or less. In summary, to improve the game experience of cloud game players, we must reduce interaction delay, and minimizing network delay is the key to achieving this.

C. Poor Stability

Due to the long distances between a cloud data center (DC) and a game player's terminal device, there is a high delay in sending video frames from the server to the player, which results in poor stability and a high potential for packet loss [5]. Poor stability can cause video scenes to freeze, which negatively impacts the player's experience.

D. Multiplayer Cloud Game Fairness

Delay skew between players is a crucial parameter for judging fairness. Overly long cloud transmission distances can lead to large latency differences, which can make it difficult for a game to provide fair service to its players, or even cause game failures due to nonplayer operations. For example, in a multiplayer first-person shooter cloud game, if Player A has high latency, the target has not yet refreshed on his screen when the other players are ready to shoot at it. Due to the high latency, Player A is unlikely to hit the target, resulting in a bad gaming experience for Player A. Huge latency differences can greatly affect the fairness of multiplayer games, reduce the user experience and ultimately lead to player attrition.

Edge computing shifts computing tasks from cloud centers to edge network. It solves the high bandwidth consumption problem of cloud gaming and meets the low latency requirements of real-time applications, while achieving high stability [6], [7]. Edge computing can effectively improve the quality of service for cloud games [8]. Yates et al. [9] introduced a new model for cloud gaming systems aimed at optimizing the timeliness of video frames based on an Age-of-Information (AoI) metric. However, compared to cloud computing, edge computing alone cannot satisfy the needs of a huge number of game players at the same time.

Therefore, this article proposes a flexible end–edge–cloud collaboration architecture based on the SDN technology, referred to as EdgeGaming-SDN, to improve cloud game performance. EdgeGaming-SDN uses a cloud DC to handle the computationally intensive part of game logic update and uses the edge to handle screen rendering tasks with extensive downlink transmission data. Combining the merits of cloud computing and edge computing, EdgeGaming-SDN increases computing power, ensures high stability, and reduces transmission delay, which then improves the service quality of cloud games.

However, the design of EdgeGaming-SDN network architecture also needs to solve several problems. In realistic scenarios, popular games usually gather a large crowd of gamers in a short period, which is more likely to cause server overload and network congestion. In particular, players usually gather in small groups, for example, in the same neighborhood or on a university campus, which adds to the server overload problem. In addition, the gathering location of players may change significantly at different times of the day. For example, there is more pressure on the servers around the peak of rush hours because many players are on subway lines during those times. Most of the available research rarely considers the mobility of gamers [10], [11]. However, when subway stops running at night, some servers around the area will be idle, resulting in a waste of resources. In realistic scenarios, channel conditions and game request areas change as some players move around, which causes an increase in the complexity of the MEC environment.

Therefore, it is challenging to jointly optimize server selection and server resource allocation in a dynamic MEC environment in order to optimize multiple objectives. Most of the existing research focuses on task and computing offloading, application layout optimization, and computing resource allocation [12], [13]. However, these approaches share a common limitation in solving practical problems: they do not consider whether the proposed algorithms cope with the ever-changing dynamic MEC environment. This article proposes a deep reinforcement learning (DRL)-based resource allocation strategy for cloud gaming to address the problem. Our design can select servers and allocate computing resources adaptively by interacting with the dynamic environment. The major contributions of this article are as follows.

1) This article proposes EdgeGaming-SDN, an SDN-assisted end–edge–cloud cloud gaming architecture, which optimizes the network architecture of cloud gaming and improves the service experience of players.

2) Based on the actual application requirements of cloud gaming, this article proposes a quality of service evaluation standard that integrates game latency, multiplayer game fairness, server computing, and network load. The architecture can provide quality assurance for user experience in cloud gaming and maintain the stability of an MEC system.
3) This article proposes a soft actor–critic (SAC)-based DRL algorithm DRLGA, which can interact with the environment in a complex and dynamic cloud game environment and make intelligent decisions. Compared with other algorithms, it can adapt to changes in the dynamic environment and meet the requirements of multimodal rewards.

The remainder of this article is presented as follows. Section II introduces the research background. Section III introduces the cloud gaming system model. Section IV introduces our problem formulation. Section V presents an adaptive resource allocation strategy. Section VI describes the experimental setup and presents the performance evaluation of the strategy. Eventually, Section VII concludes this article and discusses future work.

II. Related Work

The Quality of Experience (QoE) of cloud games is affected by many factors. Our work is to improve the QoE of users by designing a new network architecture and optimizing resource allocation. We first discuss the QoE factors that affect cloud gaming. Jarschel et al. [10] mentioned that network speed dramatically affects the user experience. With the introduction of 5G, the Internet speed has been dramatically improved, so this problem has been addressed to a large extent. Due to the low latency requirements of cloud gaming, the increase in network speed still cannot completely solve the network latency problem of cloud gaming. Geographic location may limit the degree of interaction achieved because users who are remote may experience unacceptable delays due to long distances from the DC where the service is running [11]. In addition to the effect of time delay, the image quality of a game also has a significant impact on QoE [14]. Mohammadi et al. [15] presented a hybrid graphics/video rate control method based on graphical assets for cloud gaming. The high quality of a game screen requires a high video frame rate, and the network bandwidth significantly affects the video frame rate [16], [17], [18]. The dynamic characteristics of cloud gaming network traffic also suggest that sufficient network bandwidth is necessary [19]; otherwise, low bandwidth will cause game lag and affect user service experience. Therefore, having sufficient network bandwidth can provide users with a good gaming experience. In addition, Lanzoni et al. [20] proposed that the delay variance between users is also a factor that affects user experience. Delay variance can represent the fairness between users, and fairness is also an essential factor affecting user experience [21], especially in multiplayer cloud games. There has been a lot of research into different techniques to optimize the QoE of gamers.

The MEC environment can provide users with a high-quality network environment, but the resources of MEC servers are relatively limited compared to cloud servers. In order to maximize the processing power of the edge server (ES), [22] and [23] separately studied the selective offloading of terminal computing services and the scheduling of wireless communication resources. Zhang et al. [24] studied the power control and bandwidth allocation of base stations. You et al. [25] and Xiao et al. [26] attempted to optimize the task offloading problem and the wireless resource allocation problem. Spinelli et al. [27] designed green edge gaming, which maximizes the utility of a service/infrastructure provider with time-varying edge node capacities. Cao et al. [28] also proposed an efficient optimization framework for multimedia services based on the game model, distinguished the reliability and economy of base stations according to the video service mechanism, and reduced time delay and energy consumption through wireless resource allocation strategies. Cao et al. [5] jointly investigated service placement and bandwidth allocation for MEC-enabled MCG to minimize the QoE impairment in a long time scope under a cost constraint for long-term migrations. However, the related algorithms proposed in the above papers cannot efficiently manage and control services when the number of access terminals is large and the environment changes dynamically.

The application of edge cloud technology can improve user experience to a certain extent. However, deploying services on the edge network is a very complex issue. Therefore, a control mechanism that can coordinate a distributed environment is required. SDN can have tremendous potential as a programmable network combined with edge computing. McKeown et al. proposed the concepts of OpenFlow protocol [29] and SDN [30], which separate the control plane from the data plane by decoupling hardware and software. Control signaling and business data do not affect each other, realizing flexible configuration of gateways and services. Du et al. [31] proposed a Stackelberg differential game-based cloud computing resource-sharing mechanism. Amiri et al. proposed a layered network architecture that distributes game requests to different cloud DCs to minimize service delays and improve bandwidth utilization [32].

Resource allocation strategies based on traditional methods are usually too complicated when facing massive computing services and cannot adapt well to dynamic environments. Therefore, people have hoped to solve this problem in recent years through AI technology. Min et al. [33] proposed a joint solution for ES selection and offload rate control based on CNN and RL. Chen et al. [34] proposed a computing task offloading algorithm based on dual DQN. In addition to the wireless channel quality model and energy queue state model, it also added a task queue state model. Based on the RL approach, Hao et al. [35] constructs an MDP optimization problem on the service offloading problem to minimize the long-term delay and proposed a multiple update algorithm with fast and efficient convergence. Zhang et al. [36] studied resource management schemes based on DRL. Thein et al. [37] designed an energy-saving resource allocation framework for dynamic wireless access networks

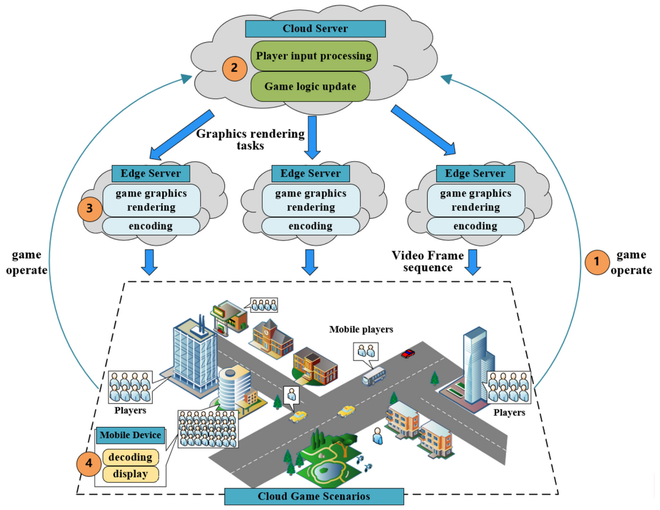


Fig. 2. End–edge–cloud cloud gaming architecture.

based on DRL. Zhang et al. [38] proposed a resource for industrial Internet of Things based on DRL. Li et al. [4] proposed reinforcement learning-based resource partitioning to improve cloud gaming responsiveness.

III. End–Edge–Cloud Collaborative System for Cloud Gaming

A. End–Edge–Cloud System Architecture

As shown in Fig. 2, multiple ESs are connected to a cloud server, while cloud gaming players are distributed around the ESs. A cloud gaming process contains four steps. First, a player sends a game request to the cloud server, which involves transmitting only a small amount of data. Second, after receiving the player's request, the cloud server runs a cloud game service to update the player's game state. A large amount of computing is needed for updating the game state. With its powerful computing resources, the cloud server can more quickly update the game state than ESs. Third, in order to transfer the graphics of the game state back to the player's device, the cloud server assigns the graphics rendering tasks to an ES. After the ES completes image rendering and compresses the image frames, it transmits them to the player's device. Giving tasks to the ES helps reduce the processing latency of game state updates and ensures the player can receive the results faster. Finally, the mobile device decodes and displays the graphics.

B. SDN-Enabled End–Edge–Cloud Resource Allocation

SDN [39] is known for its separation of data and control, in which control plane can obtain the network's real-time and global status information to achieve logical and centralized control. The control plane can deploy data forwarding policies from the network level, making the interaction between different devices in the cloud-side-end smoother. Therefore, SDN-assisted architecture can help the system handle network requests better than traditional frameworks. Therefore, this article uses the SDN technology to design the collaboration

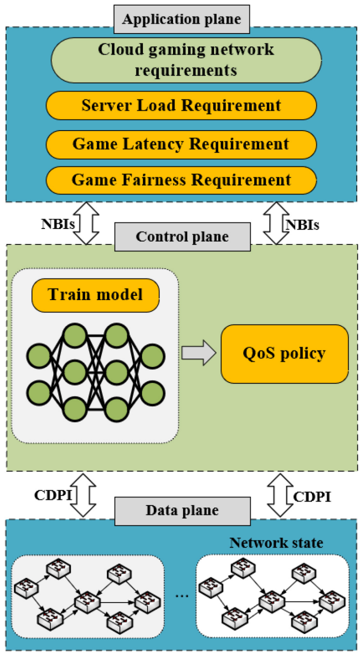


Fig. 3. SDN-enabled end–edge–cloud resource allocation.

of end–edge–cloud. As shown in Fig. 3, SDN logically consists of three planes: 1) the application plane; 2) the control plane; and 3) the data plane. The control plane communicates with the data plane using the OpenFlow protocol by control data plane interface (CDPI). The controller relays data to the application component by northbound interface (NBI). Different cloud service applications have different requirements for latency and data transfer. To reduce the latency between controllers and network devices, SDN controllers are deployed on ESs. When a new player logs in, the cloud gaming application in cloud sends the network requirements of the cloud gaming to an SDN controller via an NBI. The controller communicates with various network devices through the Control to CDPI using the OpenFlow protocol to obtain a path's current delay state and available bandwidth. A DRL agent is deployed on the SDN controller to train a resource allocation policy based on network state information. A routing decision is made for the underlying network according to the policy. Through the DRL agent in the SDN controller, the system can realize network management more efficiently and intelligently to meet the network requirements of different cloud gaming applications.

IV. Problem Formulation

In this section, we introduce the system model and problem formulation. The notations used in this article are summarized in Table I.

A. System Model

We let DC represent a cloud computing center with massive computing resources. ES represents a collection of m ESs $\{es_1, es_2, \ldots, es_m\}$. Mobile players (MPs) represents a set of n players using mobile devices $\{mp_1, mp_2, \ldots, mp_n\}$. In a real

TABLE I
SUMMARY OF THE KEY NOTATIONS

Parameter	Description (unit)
DC, ES, MP	The sets of data centers, edge servers, and players
t	The index of time slots
k	The number of time slots in a decision cycle
$T_{i,j}$	The network delay of the i^{th} player
c_i	The number of CPU cycles required by the i^{th} task
$f_{i,j}$	The CPU frequency allocated to the i^{th} task by j^{th} server
$a_{i,j}^t$	The assignment decision of i^{th} task to j^{th} server at time slot t
$T_{i,j}^e$	The computing delay of the i^{th} task on the j^{th} edge server
m, n	The number of servers and players
G	The set of players competing together in a group
S	The number of groups of players in a game
F_G	The maximum delay difference between players in group G
$l_{es_j}^{net}, l_{es_j}^{cp}$	The network and computing load of j^{th} edge server
$d_{i,j}^c, d_{i,j}^p$	The size of game tasks before game task rendering, after rendering
$r_{i,j}^e, B_{i,j}^e$	The communication rate, channel bandwidth, between the i^{th} player and the j^{th} edge server
$T_{i,j}^p$	The communication delay between the i^{th} player and the j^{th} edge server
$T_{i,j}^c, B_{i,j}^c, r_{i,j}^c$	The communication delay, channel bandwidth, communication rate between cloud server and the j^{th} edge server allocation to the i^{th} player
$p_{i,j}^c, h_{i,j}^c, \sigma_{i,j}^c$	The transmit power, channel gain, noise power from the cloud server to the j^{th} edge server allocated to the i^{th} player
$p_{i,j}^e, h_{i,j}^e, \sigma_{i,j}^e$	The transmit power, channel gain, noise power between the i^{th} player and the j^{th} edge server

cloud gaming scenario, multiple dynamic factors affect the performance of a gaming application, such as the locations of the game players, the load of the servers, wireless channel conditions, etc. For analysis, the system is discretized into a sequence of time slots $t = 0, 1, 2, \ldots, k$. The SDN controller makes service control decisions based on an intelligent allocation policy at the beginning of each time slot. We assume that each player generates a game request in a time slot. We set the time slot length to 80 ms, and any game request that cannot be completed with a time slot is considered to have timed out. Then, in the next time slot, game players will generate new game requests. The positions of some of the game players will change across different time slots. A service control decision in time slot t consists of server selection decisions a_t and server computing resource allocation decisions f_t, expressed together as $[\boldsymbol{a_t}, \boldsymbol{f_t}]$. This article aims to minimize the average latency for all MPs, ensure fairness in a multiplayer game, and balance each ES's computing and network load to improve the QoS and adaptability of the entire end–edge–cloud system. For ease of exposition, we drop time slot t in various notations in the following discussions.

Service Delay: Game latency greatly affects players' QoE, and reducing game latency is the key for game service providers to improve QoS. In this article, game latency includes the transmission delay from cloud server to ES, the processing delay of ES rendering graphics, and the transmission delay from ES to player. The traffic generated by the gamer actions will be small, on the order of 10 kb/s [40]. Therefore, the latency of a player sending requests to the cloud server is negligible. We assume that each player can only be served by a single ES in a time slot. We define a binary variable $a_{i,j}^t$ to represent the connection of the game request to the ES in time slot t. If jth server assigned to ith player, $a_{i,j}^t$ is 1. Otherwise, $a_{i,j}^t$ is 0. Then, the game latency of the ith player is calculated according to

$$T_{i,j} = T_{i,j}^c + T_{i,j}^e + T_{i,j}^p. \tag{1}$$

We explain $T_{i,j}^c$, $T_{i,j}^e$, and $T_{i,j}^p$ as follows. $T_{i,j}^c$ represents the delay from the cloud server to the jth ES. The transfer rate from the cloud server to the jth ES is defined as

$$r_{i,j}^c = B_{i,j}^c \log_2\left(1 + \frac{p_{i,j}^c h_{i,j}^c}{\sigma_{i,j}^c}\right) \tag{2}$$

where $B_{i,j}^c$ is the channel bandwidth between the cloud server and jth ES, $p_{i,j}^c$ is the transmit power from the cloud server to the jth ESs, $\sigma_{i,j}^c$ is the noise power, and $h_{i,j}^c$ is the channel gain, which is related to the communication distance.

Let $d_{i,j}^c$ represents the size of the game task transmitted from the cloud server to the ES, then the transmission delay $T_{i,j}^c$ is calculated as

$$T_{i,j}^c = \frac{d_{i,j}^c}{r_{i,j}^c}. \tag{3}$$

The delay of the ith player's task to be rendered on the jth ES is defined as

$$T_{i,j}^e = \frac{c_i}{f_{i,j}} \tag{4}$$

where c_i is the number of CPU cycles required by the ith task, and $f_{i,j}$ is the CPU frequency allocated to the task by the jth server.

The transfer rate from the jth ES to the ith player is defined as

$$r_{i,j}^e = B_{i,j}^e \log_2\left(1 + \frac{p_{i,j}^e h_{i,j}^e}{\sigma_{i,j}^e}\right) \tag{5}$$

where $B_{i,j}^e$ represents the amount of bandwidth allocated to the ith player by the jth ES, $p_{i,j}^e$ is the transmit power from the jth ES to the ith player, $\sigma_{i,j}^e$ is the noise power, and $h_{i,j}^e$ is the channel gain, which is related to the communication distance between the jth ES and ith player. When the game player's location changes, the channel gain between the player and the server changes accordingly.

The transmission delay from the jth ES to the ith player is denoted as

$$T_{i,j}^p = \frac{d_{i,j}^p}{r_{i,j}^e} \tag{6}$$

where $d_{i,j}^p$ represents the size of the game task after rendering by the ES.

Multiplayer Fairness: Players that game with each other are defined as a group $G \subset MP$. For the fairness of players in the same group, the gap in latency between the player with the lowest latency and the player with the highest latency needs to be reduced. The maximum latency difference between players in the same group is calculated as

$$F_G = \max(|T_k - T_l|) \forall mp_k, mp_l \in G. \tag{7}$$

When the value of F_G is smaller, the fairness of the system is better.

Variance of Network Resource Allocation: In real-world scenarios, games usually have a large number of concurrent requests, and at the same time, they require a large amount of bandwidth for transmission after graphics have been rendered. ESs have limited bandwidth available, and allocating many tasks to the same server for processing can significantly increase the network load on that server. This can cause network congestion, especially for some less resourceful ESs. Therefore, balancing the network load across servers is important for improving the QoS of the system. The amount of bandwidth allocated to the ith player by the jth ES is represented by $B_{i,j}$. We use a server's downstream bandwidth usage rate to define the server's network load. Then, the jth server's network load is defined as

$$l_{es_j}^{\text{net}} = \frac{\sum_{i=1}^{n} B_{i,j} a_{i,j}^{t}}{B_{es_j}} \tag{8}$$

where B_{es_j} represents the total downlink bandwidth of the ES. The variance of network resource allocation between ESs can represent the disparity of network load and it is calculated as

$$\operatorname{var}\left(l^{\text{net}}\right) = \frac{\sum_{j=1}^{m}\left(l_{es_j}^{\text{net}} - \frac{\sum_{es_j \in ES} l_{es_j}^{\text{net}}}{m}\right)^2}{m}. \tag{9}$$

Variance of Computing Resource Allocation: As the number of tasks per server varies and the ESs do not have the same computing capacity, different ESs have different computing loads. If too many graphics rendering tasks are allocated to ESs with low computing capacity, the ESs can be easily overloaded. To adequately measure the computing resource usage of different ESs, we define the computing load of the jth ES as the CPU computing resource usage, denoted as

$$l_{es_j}^{cp} = \frac{\sum_{i=1}^{n} f_{i,j} a_{i,j}^{t}}{O_{es_j}} \tag{10}$$

where O_{es_j} represents all the computing resources of the jth ES's CPU, and $f_{i,j}$ denotes the computing resources allocated by the jth ES to the ith player. Then, we use the variance of computing load to express the load differences among ESs. It can be calculated as

$$\operatorname{var}\left(l^{cp}\right) = \frac{\sum_{j=1}^{m}\left(l_{es_j}^{cp} - \frac{\sum_{es_j \in ES} l_{es_j}^{cp}}{m}\right)^2}{m}. \tag{11}$$

B. Problem Formulation

1) Balance Edge Servers Network and Computing Load: To ensure the stability of the cloud gaming system, the network load between servers needs to be balanced. The network load balancing problem can be formulated as a constrained optimization problem whose objective is to minimize the network load variance among servers, subject to the bandwidth capacity of the servers

Objective 1:

$$\min_{a,f}\left(\operatorname{var}\left(l^{\text{net}}\right)\right)$$

Subject to:

$$\begin{aligned}
&i. \quad \sum_{j=1}^{m} a_{i,j}^{t} = 1 \ \forall i \in \{1,\ldots,n\}\\
&ii. \quad a_{i,j}^{t} \in \{0,1\} \ \forall j \in \{1,\ldots,m\} \ \forall i \in \{1,\ldots,n\}\\
&iii. \quad \sum_{i=1}^{n} B_{ij} a_{i,j}^{t} \le B_{es_j} \ \forall j \in \{1,\ldots,m\}.
\end{aligned} \tag{12}$$

Constraints (i) and (ii) ensure that $a_{i,j}^{t}$ is a binary variable and only one ES serves a player. Constraint (iii) ensures that the sum of the downlink bandwidth allocated to all game requests rendered by an ES does not exceed the downlink bandwidth of the ES.

In addition to maintaining a balanced network load, it is also essential to keep a balanced computing load between servers. It is necessary to avoid crashes due to overloaded servers and wasted resources due to overly idle servers. Recall that we define $\operatorname{var}(l^{cp})$ to evaluate the computing load balance between servers. The computing load balance problem can be formulated as an optimization problem, shown as follows:

Objective 2:

$$\min_{a,f}\left(\operatorname{var}\left(l^{cp}\right)\right)$$

Subject to (i), (ii) and:

$$\begin{aligned}
&iv. \quad f_{i,j} \in \left[0, O_{es_j}\right] \ \forall j \in \{1,\ldots,m\} \ \forall i \in \{1,\ldots,n\}\\
&v. \quad \sum_{i=1}^{n} f_{ij} a_{i,j}^{t} \le F_{es_j} \ \forall j \in \{1,\ldots,m\}.
\end{aligned} \tag{13}$$

Except that Constraints (i) and (ii) are the same as those defined in (12). Constraint (iv) represents the CPU frequency assigned to the ith player by the jth server, which takes a value between 0 and O_{es_j}. Constraint (v) guarantees that the sum of the CPU frequencies allocated to the players on each server does not exceed the maximum CPU frequency of the server.

2) Minimize Average Latency Experienced by Players: As discussed before, $T_{i,j}$ represents the service latency of player i. We aim to minimize the average service latency of n game players by selecting servers and allocating computing resources to the selected servers. Thus, the average service latency minimization problem can be defined as

Objective 3:

$$\min_{a,f}\left(\frac{\sum_{i=1}^{n}\sum_{j=1}^{m} T_{i,j} a_{i,j}^{t}}{n}\right)$$

Subject to (i, ii, iii, iv, v) and:

$$vi. \quad T_{i,j} \le T_{\max} \ \forall j \in \{1,\ldots,m\} \ \forall i \in \{1,\ldots,n\}. \tag{14}$$

$T_{\max}$ denotes the maximum acceptable delay for the players. Constraint (vi) indicates that the latency per game player cannot exceed the maximum tolerable latency for the player.

3) Maximize Fairness Between Players: Fairness is crucial for players competing in the same group in multiplayer games. According to (7), we use the maximum latency difference between players in the same group to measure fairness. When n game players send their game requests, they are divided into S groups according to the prescribed number of players per group for multiplayer games. $G = G_1 \cup \cdots \cup G_S,\ G_i \cap G_j = \emptyset$. The fairness of the n players can then be expressed as the mean of the latency differences between the players of the S groups. We aim to maximize the overall fairness of all n players, i.e., to reduce the mean latency differences across the groups. Thus, maximizing system fairness is essentially the following minimization problem:

$$\begin{aligned}&\text{Objective 4:}\\ &\qquad\min_{a,f}\left(\frac{\sum_{i=1}^{S} F_{G_i}}{S}\right)\\ &\text{Subject to: } i, ii, iii, iv, v, vi.\end{aligned}\tag{15}$$

4) Joint Optimization Formulation: Based on the above analysis, we have formulated four optimization problems. However, the optimal solutions obtained from these four optimization problems are not necessarily compatible. Therefore, we propose a multiobjective joint optimization formulation to make server selection and computing resource allocation policies. We introduce a cloud gaming utility function U_t with weight coefficients w_1, w_2, w_3, and w_4 assigned to the four optimization objectives, where $w_1 + w_2 + w_3 + w_4 = 1$. The utility function is defined as follows:

$$\begin{aligned}U_t = T_{\max} - w_1 \cdot \left(\operatorname{var}\left(l^{\text{net}}\right)\right) - w_2 \cdot \left(\operatorname{var}\left(l^{cp}\right)\right)\\ - w_3 \cdot \left(\frac{\sum_{i=1}^{n}\sum_{j=1}^{m} T_{i,j} a_{i,j}^t}{n}\right) - w_4 \cdot \left(\frac{\sum_{i=1}^{S} F_{G_i}}{S}\right).\end{aligned}\tag{16}$$

To solve for an optimal control decision, including server selection $\boldsymbol{a_t} = (a_{i,j}^t)_{m\times n}$ and the computing resource allocation $\boldsymbol{f_t} = (f_{i,j}^t)_{m\times n}$ to maximize U_t, a mixed-integer nonlinear programming problem P_1 is constructed as follows:

$$\begin{aligned}&P_1 = \max_{a,f} U_t(d_t, c_t, B_t, \sigma_t, \boldsymbol{l_t}, \boldsymbol{a_t}, \boldsymbol{f_t})\\ &\text{subject to:}\\ &C1: U_t = T_{\max} - w_1 \cdot \left(\operatorname{var}\left(l^{\text{net}}\right)\right) - w_2 \cdot \left(\operatorname{var}\left(l^{cp}\right)\right)\\ &\qquad -w_3 \cdot \left(\frac{\sum_{i=1}^{n}\sum_{j=1}^{m} T_{i,j} a_{i,j}^t}{n}\right) - w_4 \cdot \left(\frac{\sum_{i=1}^{S} F_{G_i}}{S}\right)\\ &C2: \sum_{j=1}^{m} a_{i,j}^t = 1\ \forall i \in 1, \ldots, n\\ &C3: a_{i,j}^t \in 0, 1\ \forall j \in 1, \ldots, m\ \forall i \in 1, \ldots, n\\ &C4: \sum_{i=1}^{n} B_{ij} a_{i,j}^t \le B_{es_j}\ \forall j \in 1, \ldots, m\\ &C5: f_{i,j} \in \left[0, O_{es_j}\right]\ \forall j \in 1, \ldots, m\ \forall i \in 1, \ldots, n\\ &C6: \sum_{i=1}^{n} f_{ij} a_{i,j}^t \le F_{es_j}\ \forall j \in 1, \ldots, m\\ &C7: T_{i,j} \le T_{\max}\ \forall j \in 1, \ldots, m\ \forall i \in 1, \ldots, n.\end{aligned}\tag{17}$$

In a multiplayer game, player's graphics are roughly the same. Therefore, $B_{i,j}$ is set to the same value for different players in a game, and it can vary according to different categories of games. $\boldsymbol{l_t} = (l_1, l_2, \ldots, l_n; l_{e1}, l_{e2}, \ldots, l_{em})$ is the set of locations of n players and m ESs. Since environmental parameters in practical systems are dynamic and uncertain, solving the optimal decision for a certain time slot is not globally optimal. In addition, decisions within the current time slot may also affect the future environmental state. Therefore, it is necessary to consider the utility of a decision within the current time slot and the long-term utility in future. We introduce a discrete-time Markov decision process to model the system. To achieve better adaptive control, we treat a decision cycle containing k consecutive time slots as an event episode. Based on P_1, a mixed-integer nonlinear programming problem P_2 with Markov properties is constructed, with the total utility within an event episode as the optimization objective, and it is shown in

$$\begin{aligned}&P_2 = \max_{a,f} \sum_{t=1}^{k} U_t(d_t, c_t, B_t, \sigma_t, \boldsymbol{l_t}, \boldsymbol{a_t}, \boldsymbol{f_t})\\ &\text{subject to}: C1, C2, C3, C4, C5, C6, C7\end{aligned}\tag{18}$$

where d_t represents the size of the game task before rendering. c_t is the number of CPU cycles required by rendering task. B_t represents the bandwidth needed by one rendering task. The channel gain changes as the position changes from time slot t to time slot $t+1$. σ_t is the noise power. $\boldsymbol{a_t}$ and $\boldsymbol{f_t}$ represent the server selection decision and the computing resource allocation decision, respectively.

V. Resource Allocation via Deep Reinforcement Learning

We apply DRL algorithms to solve the optimization problems introduced in the previous section. Considering the continuous action space and the complex optimization objective, we propose an adaptive resource allocation strategy based on SAC [41] to solve the optimal server selection and server resource allocation problem introduced in Section VI. In the following, we first give an introduction to SAC algorithm.

A. Reinforcement Learning via SAC

Reinforcement learning is how an agent interacts with its environment and continuously improves its strategy π according to the rewards obtained. Reinforcement learning can be divided into value-based and policy-based methods. Q-learning is a classical value-based reinforcement learning algorithm. Q-learning records the updated value of each action-value function at time step t through a Q-table.

However, when the action space is large or continuous, the approximation of the action-value function will become problematic. A policy gradient-based reinforcement learning algorithm does not need to estimate the action-value function and can learn the optimal policy π^* directly. SAC learns a policy π to maximize both rewards and entropy such that

$$\pi^* = \operatorname*{argmax}_{\pi} \sum_{t} \mathbb{E}_{(s_t, a_t)\sim\rho_\pi}\left[r(s_t, a_t) + \alpha\mathcal{H}(\pi(. \mid s_t))\right].\tag{19}$$

The entropy is defined as

$$\mathcal{H}(\pi(. \mid s_t)) = \mathbb{E}_{a\sim\pi(.|s_t)}\left[-\log(\pi(a \mid s_t))\right]\tag{20}$$

where s_t and a_t represent the current state and action, respectively. coefficient α is temperature coefficient, representing the weight of entropy. $\rho_\pi(s_t)$ and $\rho_\pi(s_t, a_t)$ are denoted as the state and state–action marginals of the trajectory distribution induced by a policy π. The reward keeps changing during the training process, so using a fixed temperature α will lead to instability throughout the training. Therefore, we use a temperature α that can be adjusted automatically. The initial value of temperature α is random. When the agent does not find the optimal action, we increase the temperature α to explore. Temperature α will gradually decrease when the optimal action is determined. When target entropy is $\mathcal{H}_0$, the loss of temperature α is as follows:

$$J(\alpha) = \mathbb{E}_{a_t \sim \pi_t}\left[-\alpha \log \pi_t(a_t \mid \pi_t) - \alpha \mathcal{H}_0\right]. \tag{21}$$

Compared with deterministic policy algorithm, such as deep deterministic policy gradient (DDPG) [42], the SAC algorithm needs to maximize entropy in the process of exploration. Therefore, the SAC algorithm has a stronger exploration ability and is more likely to achieve better results for multimodal rewards.

The SAC algorithm uses an actor–critic structure. $\mathcal{D}$ denotes a replay buffer which is used to store the historical experience. An actor contains a policy function $\pi_\phi(a_t|s_t)$, which is responsible for generating actions and interacting with the environment. A critic uses a value function $Q_\theta(s_t, a_t)$ to evaluate the actor's performance and to guide the actor's actions in the next phase. The critic's parameters θ can be learned by minimizing

$$J_Q(\theta) = \mathbb{E}_{(s_t, a_t, s_{t+1}) \sim \mathcal{D}}\Big[\big(Q_\theta(s_t, a_t) - (r(s_t, a_t) + \gamma V_\theta(s_{t+1}))\big)^2\Big] \tag{22}$$

where

$$V_\theta(s_{t+1}) = \mathbb{E}_{a_t \sim \pi_\phi(.|s_{t+1})}\left[Q_\theta(s_{t+1}, a_t) - \alpha \log\big(\pi_\phi(a_t \mid s_{t+1})\big)\right]. \tag{23}$$

The policy's parameters can be learned by minimizing

$$J_\pi(\phi) = \mathbb{E}_{s \sim \mathcal{D}}\left[\mathbb{E}_{a_t \sim \pi_\phi(.|s_t)}\left[\alpha \log \pi_\phi(a_t \mid s_t) - Q_\theta(s_t, a_t)\right]\right]. \tag{24}$$

We use a target critic with weight $\bar{\theta}$, which is obtained through an exponentially moving average of the parameter θ

$$\bar{\theta} \leftarrow (1 - \eta)\bar{\theta} + \tau\theta \tag{25}$$

where η is the soft update factor.

We use Q_{θ_1} and Q_{θ_2} to represent two critics and compute the TD-target as the minimum to reduce the over-estimation bias. Then, the TD-target becomes

$$y_t = r + \gamma \mathbb{E}_{a_t \sim \pi(|s_{t+1})}\left[\min_{i=1,2} Q_{\bar{\theta}_i}(s_{t+1}, a_t) - \alpha \log(\pi(a_t \mid s_{t+1}))\right]. \tag{26}$$

And the losses

$$J_Q(\theta) = \mathbb{E}_{(s_t, a_t, s_{t+1}) \sim \mathcal{D}}\Big[\big(Q_{\theta_1}(s_t, a_t) - y_t\big)^2 + \big(Q_{\theta_2}(s_t, a_t) - y_t\big)^2\Big] \tag{27}$$

$$J_\pi(\phi) = \mathbb{E}_{s \sim \mathcal{D}}\left[\mathbb{E}_{a_t \sim \pi_\phi(|s_t)}\left[\alpha \log \pi_\phi(a_t \mid s_t) - \min_{i=1,2} Q_{\bar{\theta}_i}(s_{t+1}, a_t)\right]\right]. \tag{28}$$

B. State Space

Recall that the end–edge–cloud architecture is introduced in Section III, shown in Figs. 2 and 3. In time slot t, the SDN controller collects the state of the network environment for a DRL agent to make decisions. At the next time slot $t+1$, the channel gain of a user will change with the user's new location. The user's task size, the number of CPU cycles required, and the bandwidth between the user and ESs vary from time to time. The environmental state $\boldsymbol{S_t}$ within the time slot t can be defined as a set containing five key pieces of environmental information, denoted as follows: $\boldsymbol{S_t} = (d_t, c_t, B_t, \sigma_t, \boldsymbol{l_t})$, where d_t represents all players' rendering tasks' data size; c_t represents the number of CPU cycles required to process the graphics rendering tasks; B_t represents the downlink bandwidth required for tasks; σ_t is the noise power; and vector $\boldsymbol{l_t}$ represents the locations of ESs and players.

C. Action Space

The action space is divided into two parts: 1) server selection decision and 2) computing resource allocation decision. The former assigns the rendering tasks to the appropriate ESs and the latter allocates computing resources. The server selection decision determines which servers will be connected to which rendering tasks in time slot t. If $a_{i,j}^t = 1$, the ith rendering task will be assigned to the jth server in the time slot t. The computing resource allocation decision determines the number of computing resources that will be allocated to the ith task by the ES if the task is assigned to the server. Thus, the resource allocation decision within time slot t can be expressed as $f_{i,j}^t \in [0, O_{es_j}]$. Action A_t within time slot t is denoted by

$$A_t = [\boldsymbol{a_t}, \boldsymbol{f_t}]. \tag{29}$$

D. Reward Function

After making a decision based on state, the agent is rewarded with R_t. The utility U_t given in (16) for time slot t can be used to define the reward function for the time slot t, i.e.,

$$R_t = U_t. \tag{30}$$

Following the standard RL approach, the reward function R within an episode is given as follows:

$$R = \mathbb{E}\left[\sum_{t=1}^{k} \gamma^t \cdot R_t\right] \tag{31}$$

where γ is a discount factor.

We propose a DRL-based cloud gaming resource allocation algorithm (DRLGA), shown in Fig. 4. The algorithm contains three parts, i.e., an actor, a critic, and a replay memory D. The actor maps states to actions, the critic estimates values of states and state–action pairs, and the replay memory is used to store experience. The two value networks each consists of fully

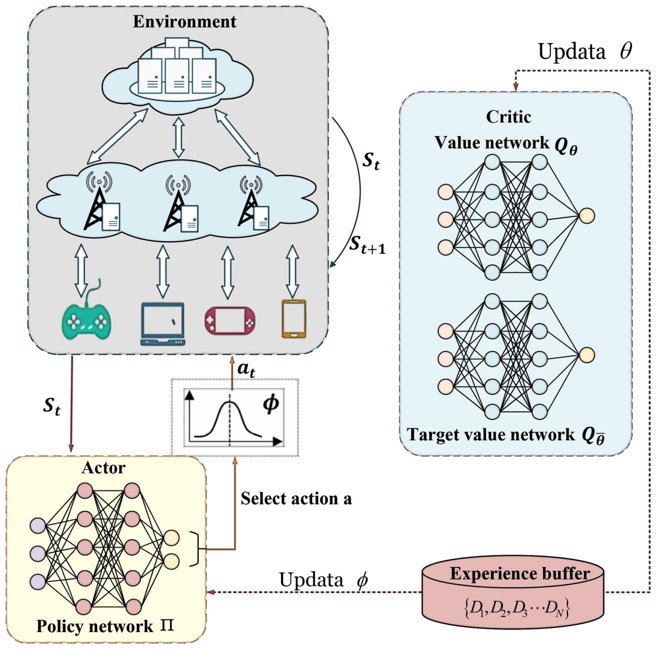


Fig. 4. SAC framework.

connected layers, including two hidden layers with 256 nodes. The learning process alternates between experience collection from the environment with the current policy and updating the parameters of the function approximators based on batches sampled from the replay memory. The details of the algorithm are shown in Algorithm 1.

Algorithm 1 DRLGA

```
Input: Initialize the weights of DRLGA's DNN: $\theta_1$, $\theta_2$, $\phi$;
    Initialize target network weights: $\bar{\theta}_1 \leftarrow \theta_1, \bar{\theta}_2 \leftarrow \theta_2$;
    Initialize an empty replay pool: $\mathcal{D} \leftarrow \varnothing$;
 1: for each episode do
 2:     Reset the agent and the simulation environment of proposed resource allocation model;
 3:     Reset the initial state $S_0$, including the size of tasks, the location of players, the noise power and the bandwidth state; reset the initial reward $R_0 = 0$;
 4:     for $[t = 1, 2, \ldots, k]$ do
 5:         Sample action from the policy: $\mathbf{a}_t \sim \pi_\phi(\mathbf{a}_t \mid \mathbf{s}_t)$
 6:         Observe next state $s_{t+1}$, $R_t$ from the environment and determine whether $s_{t+1}$ is the terminal;
 7:         Update replay memory:
            $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}_t, \mathbf{a}_t, \mathbf{r}_t, \mathbf{s}_{t+1})\}$
 8:     end for
 9:     for each gradient step do
10:         Update the Q-function parameter:
            $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i)$ for $i \in \{1, 2\}$
11:         Update policy weights:
            $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$
12:         Adjust temperature:
            $\alpha \leftarrow \alpha - \lambda \hat{\nabla}_\alpha J(\alpha)$
13:         Update target network weights:
            $\bar{\theta}_i \leftarrow \eta\theta_i + (1 - \eta)\bar{\theta}_i$ for $i \in \{1, 2\}$
14:     end for
15: end for
Output: $\theta_1, \theta_2, \phi$
```

VI. System Evaluation

This section describes our simulation setup, presents the experimental scenario, and briefly introduces existing algorithms we used in our comparisons. We also provide the performance evaluation of the proposed optimization method and discuss its effectiveness for both gaming service providers and gamers.

A. Experimental Settings

To evaluate our proposed method in practical scenarios, we use the Kaist/WiBro data set [43]. The data set collects CBR and VoIP traffic from the WiBro network on Seoul subway train line 6 for network performance analysis. The maximum speed of the Seoul subway is 90 km/h, which can reasonably simulate the dynamic changes of an MEC system in the scenario of high-speed movement. In addition, subway commuters are more likely to use mobile devices for gaming than those traveling by other transportation modes. Therefore, we use the location information of commuters in the Kaist/WiBro data set to simulate the changes of players' locations in a cloud game system. The location trajectories of some gamers are visualized in Fig. 5. As shown in the figure, gamers are always on the move and some users have similar trajectories, which indicates the gathering characteristics of gamers. To better simulate gamer surge in a short period, we use the movement trajectories of 100 users for simulation experiments, assuming that 100 MPs are simultaneously playing the same

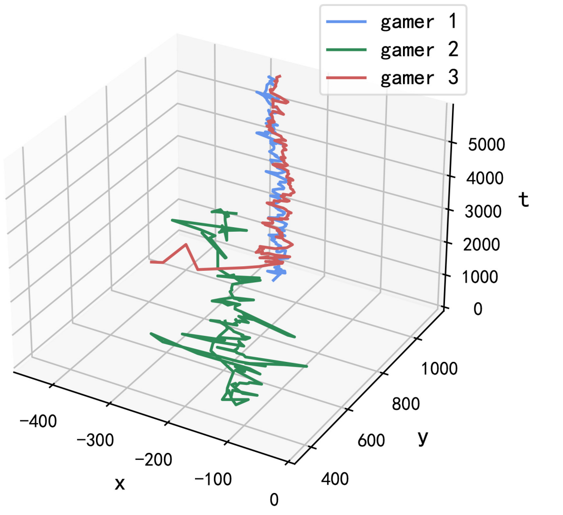


Fig. 5. 3-D gamer location track map.

multiplayer cloud game. Ten ESs are deployed to cover the 100 players' locations evenly in our simulated environment. The cloud service is deployed at a slightly distant site outside the area. The channel gain is set to $h_{i,j} = d_{i,j}^{-\varsigma}$. $d_{i,j}$ is the distance between the ith gamers and the jth server, and we set $\varsigma = 3$ [44]. We simulate the traffic variation characteristics of the first-person shooter game half-life. The task data size sent down from the cloud server to the ESs is uniformly distributed

TABLE II
SIMULATION PARAMETERS

Parameter	Description (unit)	value
NUM_{D}	Experience pool size	100000
l_r	Initial learning rates	0.001
n_b	Mini_batch size	256
γ	Discount factor	0.99
η	Soft update factor	0.005
α_0	Initial entropy coefficient	0.8
e	Number of training episodes	280

in (3 kb, 4 kb), the rendered data size is uniformly distributed in (90 kb, 120 kb), and the downlink bandwidth is uniformly distributed in (80 Mb/s, 100 Mb/s). The computing power of an ES is set to $O_{es_j} = 2$ GHz/s, and the number of CPU cycles required for the rendering process of the task is uniformly distributed in (100 MHz, 200 MHz). The hyperparameters used for the training in our DRL are given in Table II.

To compare our algorithm with existing algorithms, we look at the following four classic algorithms based on continuous action space: 1) DDPG; 2) proximal policy optimization (PPO); 3) advantage actor–critic (A2C); and 4) twin-delayed deep deterministic (TD3) policy gradient algorithm. In addition, we also look at two traditional methods of selecting servers: 1) Random and 2) Min_dist.

1) PPO [45] is currently one of the mainstream DRL algorithms. It is suitable for both discrete and continuous control and has achieved good results. However, PPO faces a serious sample inefficiency problem and requires many samples to learn, which is unacceptable for training in a natural environment.
2) DDPG [42] is a deterministic policy gradient algorithm in continuous action space that selects an optimal action during the training process. DDPG overcomes the low sample efficiency problem of the PPO algorithm, but it is easy to fall into local optimal solutions in some cases.
3) A2C [45] provides an actor–critic architecture that is able to learn parameterized policies. A2C multiprocess can make full use of computing resources and improve learning speed.
4) TD3 [46] is also a continuous action space DRL algorithm based on AC architecture. It uses twin networks to represent different Q values. The persistent overestimation is suppressed by selecting the smallest one as the target Q value.
5) The Random algorithm is to assign tasks to random by selected servers. Its decision making is fast, but it is easy to cause high latency due to unreasonable assignments.
6) Min_dist is a minimum distance offloading algorithm that assigns the geographically closest server to each gamer, which can be close to an optimal allocation scheme if the server resources are sufficient.

B. Analysis of Results

This section analyzes the performance of our algorithm and other different DRL algorithms and traditional algorithms for cloud games in the same environment. The performance metrics include the average delay of players, the delay difference between the same group of players in a multiplayer game, and network and computing load. In addition, we will analyze the performance of the DRLGA algorithm in different MEC environments.

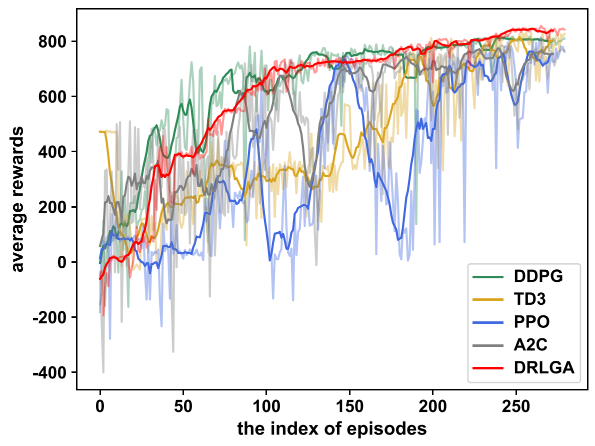


Fig. 6. Convergence comparison of different algorithms.

1) Training Performance: DRLGA's learning curve is shown in Fig. 6. Each color represents an algorithm, and for each algorithm, the darker color curve indicates the average reward value of three adjacent episodes, and the light curve indicates the reward value of each episode. The reward of DRLGA becomes larger as the number of episodes increases, indicating that the iterative training process improves the algorithm's performance. As shown in Fig. 6, the reward values of all algorithms are relatively small in the early stages of training, and the performance is poor because not enough information is learned to make decisions. The rewards gradually reach smooth and stable values as the number of episodes increases. Compared with other algorithms, the DRLGA algorithm is faster to train and does not easily fall into local optimal solutions. The DRLGA algorithm uses a replay memory to store the experience generated in the learning process and randomly selects samples while training DNNs. This breaks the temporal correlations between samples and improves sample efficiency, allowing the DRLGA algorithm to converge quickly. PPO and A2C are on-policy methods with poor sample efficiency, requiring very high sample sizes and sample complexity. In the complex scenario of multimodal rewards in cloud gaming, it is more challenging to balance exploration and exploitation, and the exploration capability is lower than the SAC algorithm. The DDPG algorithm also has an experience replay mechanism. However, compared to DRLGA, the training process of DDPG does not consider maximizing entropy, making the algorithm susceptible to locally optimal solutions. The DRLGA algorithm adds the goal of maximizing entropy, which enhances the exploration ability of the algorithm and reduces the training time while increasing the reward.

2) Average Latency: All algorithms are set with default parameters, and the number of players is set to 20, 40, 60, 80, and 100, respectively. Fig. 7 shows the DRLGA algorithm's latency performance compared with other algorithms for different number of users. The average latency of players

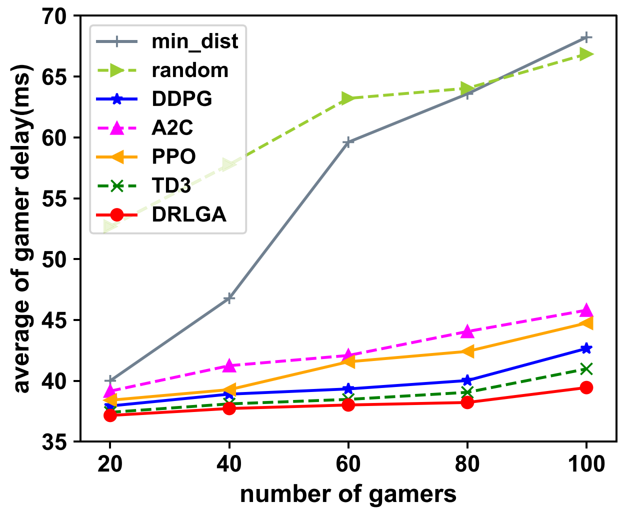


Fig. 7. Average game player latency.

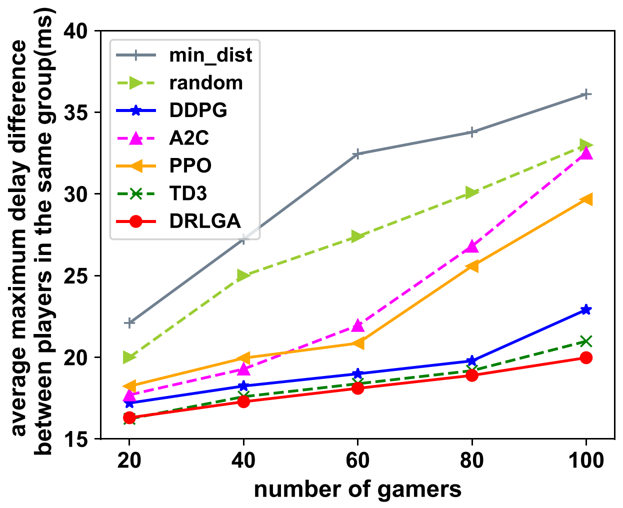


Fig. 8. Average maximum delay difference between players in the same group.

increases with the increase of the number of users. It shows that the rise in the number of players brings challenges to the MEC system, and players' experience is easily affected when the number of users reaches a large number. We see that the average latency of the DRLGA algorithm is lower than other classical reinforcement learning algorithms. In addition, when the number of players increases, the DRLGA algorithm has an obvious optimization effect compared with the traditional algorithm and can better adapt to a dynamic environment. For example, when the number of gamers was increased to 100, the average player latency of players for DRLGA was 39.4 ms, a 42.25% reduction in latency over the traditional Min_dist algorithm and a 3.8% reduction over the suboptimal TD3 algorithm. When the number of players is small, the average latency of the Min_dist algorithm is close to that of the DRLGA algorithm, but as the number of players grows, the average player latency of the Min_dist algorithm grows rapidly. Because the computing resources are sufficient when the number of players is small, choosing the server closest to the user to handle the rendering task can reduce the latency of graphic transmission. However, as the number of players grows, some users may gather in the same area, and the Min_dist algorithm assigns the rendering tasks of the gathered players to the servers closest to them. This causes the selected server to allocate only a small amount of computing resources to each player, so the task processing latency increases significantly.

3) Fairness Performance Analysis: Fig. 8 indicates the average value of the maximum delay difference between players in the same group for different algorithms with different numbers of players. Fig. 8 indicates that the DRLGA algorithm still has a clear advantage, and it can control the latency within 20 ms. According to the persistence of vision, a human can hardly perceive the delay within 20 ms on the screen. Therefore, the DRLGA algorithm can ensure the fairness of multiplayer games. The average maximum delay difference between players in the same group for the traditional algorithms increase sharply with the number of users and cannot keep fairness in multiplayer games.

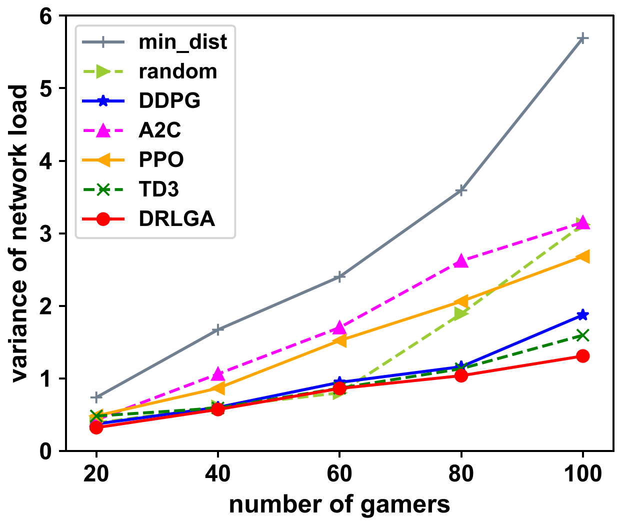


Fig. 9. Network load balancing.

4) Computation and Network Load Analysis: Our experiments are performed with default parameters. Fig. 9 shows that the DRLGA algorithm has the smallest values and has a slight variation in slope as the number of players increases. It shows that the DRLGA algorithm always obtains a relatively stable and low network load variance as the number of players increases. The DRLGA algorithm fully considers the current network load state and fully uses each ES's resources through server selection and resource allocation. It effectively prevents the increase in player service delay caused by network congestion. The advantages of the DRLGA algorithm for the multimodal reward problem are again demonstrated. As shown in Fig. 9, value of Min_dist is large, and the slope of the fold becomes larger as the number of players increases. Min_dist leads to a server imbalance in network load due to the aggregation of players, with some servers overloaded and some server resources wasted. For further verification, we analyze the allocation of computing resources under different algorithms shown in Fig. 10. As the number of players increases,

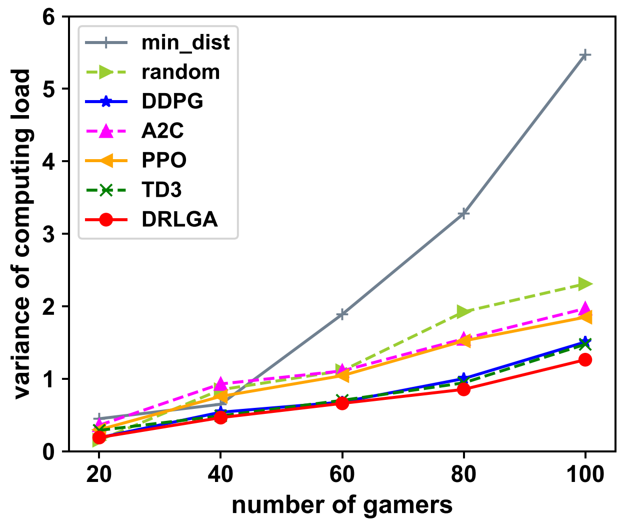


Fig. 10. Computing load balancing.

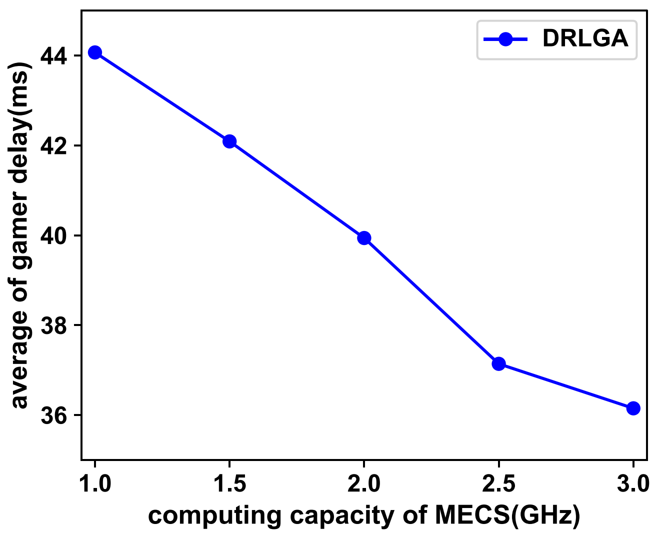


Fig. 11. Comparison of average latency of gamers using servers of different computing power.

players demand more and more computing resources from a server. Moreover, due to the mobility and aggregation of players, the demand for computing resources becomes more and more complex. The DRLGA algorithm can maintain the computing load difference between servers at a relatively stable level through resource allocation, thus maintaining the stability of the whole MEC environment.

We analyze the impact of different environments on DRLGA, such as the impact of different computing power on the average latency of players. The experimental parameters are set to default values, and the computing power of an ES is set to 1.0, 1.5, 2.0, 2.5, and 3.0 GHz. As shown in Fig. 11, the average delay of players significantly decreases as the computing power of the ES increases. However, when the computing power increases to 2.5 GHz, the rate of average latency reduction slows down. Therefore, we can select the most suitable server device based on this feature.

In addition, we examine the effect of weighting factors in the utility on the system's performance in the same

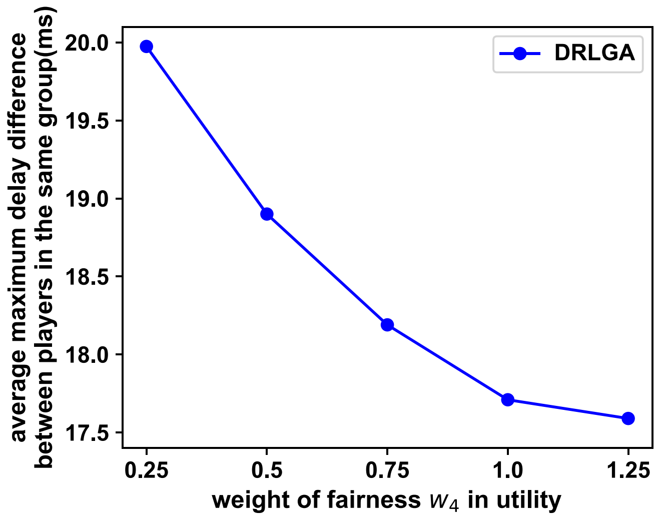


Fig. 12. Comparison of the average of the maximum delay difference between players in the same group with changing fairness weight coefficient.

environment. As shown in Fig. 12, keeping the MEC environment parameters settings constant, the value of w_4 is set to ε, $w_1 = w_2 = w_3 = (1 - \varepsilon)/3$. The values of ε are assumed to be 0.25, 0.5, 0.75, 1.0, and 1.25, respectively. As the fairness weighting factor increases, the average of the maximum delay difference between players in the same group shows a decreasing trend, which means that the system becomes fairer. Therefore, the weighting coefficients of service performance can be adjusted according to the characteristics of different games. For example, the fairness coefficient of multiplayer competitive games can be set higher than that of multiplayer social games.

VII. Conclusion

This article proposed an architecture that applies SDN technology to the joint optimization of server selection and computing resource allocation in dynamically changing cloud gaming scenarios, named EdgeGaming-SDN. The optimization problem is considered in three aspects according to the particular requirements of real scenarios of cloud gaming: 1) reducing the latency of cloud gaming services; 2) balancing the server computation and network load; and 3) improving the fairness of multiplayer games. We propose a DRLGA algorithm that can effectively solve the problems, such as the surge in the number of requests and the change of player locations according to the dynamic changes of network conditions while optimizing various performances. Although our work can improve resource utilization through intelligent allocation, it does not consider the reuse of resources. There is a bottleneck in performance improvement with limited resources. In the future, we will consider incorporating caching mechanisms to improve the reuse of computing resources. In addition, this article does not consider the further optimization of communication resources. Future work could consider the multiplexing of spectrum resources based on nonorthogonal multiple access techniques.

Acknowledgment

The authors would like to thank the Editor-in-Chief, the Associate Editor, and the reviewers for their insightful comments and suggestions.

References

[1] Y. Zhang and Y. Zhang, "Discussion on key technologies of cloud game based on 5G and edge computing," in *Proc. IEEE 20th Int. Conf. Commun. Technol. (ICCT)*, 2020, pp. 524–527.

[2] S. F. Lindström, M. Wetterberg, and N. Carlsson, "Cloud gaming: A QoE study of fast-paced single-player and multiplayer gaming," in *Proc. IEEE/ACM 13th Int. Conf. Utility Cloud Comput. (UCC)*, 2020, pp. 34–45.

[3] China Academy of Information and Communications Technology and 5G Industry Alliance, "Research report on key technologies of cloud games," CAICT, Beijing, China, White Paper, 2021.

[4] Y. Li et al., "Reinforcement learning-based resource partitioning for improving responsiveness in cloud gaming," *IEEE Trans. Comput.*, vol. 71, no. 5, pp. 1049–1062, May 2022.

[5] T. Cao, Z. Qian, K. Wu, M. Zhou, and Y. Jin, "Service placement and bandwidth allocation for MEC-enabled mobile cloud gaming," in *Proc. IEEE 22nd Int. Symp. World Wireless Mobile Multimedia Netw. (WoWMoM)*, 2021, pp. 179–188.

[6] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 4, pp. 30–39, Jan. 2017.

[7] W. Shi, C. Jie, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016.

[8] X. Zhang et al., "Improving cloud gaming experience through mobile edge computing," *IEEE Wireless Commun.*, vol. 26, no. 4, pp. 178–183, Aug. 2019.

[9] R. D. Yates, M. Tavan, Y. Hu, and D. Raychaudhuri, "Timely cloud gaming," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, 2017, pp. 1–9.

[10] M. Jarschel, D. Schlosser, S. Scheuring, and T. Hoßfeld, "An evaluation of QoE in cloud gaming based on subjective tests," in *Proc. 5th Int. Conf. Innov. Mobile Internet Serv. Ubiquitous Comput.*, 2011, pp. 330–335.

[11] T. Hobfeld, R. Schatz, M. Varela, and C. Timmerer, "Challenges of QoE management for cloud applications," *IEEE Commun. Mag.*, vol. 50, no. 4, pp. 28–36, Apr. 2012.

[12] L. Zhao, J. Wang, J. Liu, and N. Kato, "Optimal edge resource allocation in IoT-based smart cities," *IEEE Netw.*, vol. 33, no. 2, pp. 30–35, Mar./Apr. 2019.

[13] H. Lin, S. Zeadally, Z. Chen, H. Labiod, and L. Wang, "A survey on computation offloading modeling for edge computing," *J. Netw. Comput. Appl.*, vol. 169, Nov. 2020, Art. no. 102781.

[14] M. Jarschel, D. Schlosser, S. Scheuring, and T. Hoßfeld, "Gaming in the clouds: QoE and the users' perspective," *Math. Comput. Model.*, vol. 57, nos. 11–12, pp. 2883–2894, 2013.

[15] I. S. Mohammadi, M. Ghanbari, and M. R. Hashemi, "A hybrid graphics/video rate control method based on graphical assets for cloud gaming," *J. Real-Time Image Process.*, vol. 19, no. 1, pp. 41–59, 2022.

[16] Y. Lin and H. Shen, "CloudFog: Leveraging fog to extend cloud gaming for thin-client MMOG with high quality of service," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 2, pp. 431–445, Feb. 2017.

[17] W. Cai, V. C. M. Leung, and L. Hu, "A cloudlet-assisted multiplayer cloud gaming system," *Mobile Netw. Appl.*, vol. 19, no. 2, pp. 144–152, 2014.

[18] S. Zadtootaghaj, S. Schmidt, and S. Möller, "Modeling gaming QoE: Towards the impact of frame rate and bit rate on cloud gaming," in *Proc. 10th Int. Conf. Qual. Multimedia Exp. (QoMEX)*, 2018, pp. 1–6.

[19] W. Cai and V. C. M. Leung, "Decomposed cloud games: Design principles and challenges," in *Proc. IEEE Int. Conf. Multimedia Expo Workshops (ICMEW)*, 2014, pp. 1–4.

[20] T. Lanzoni, G. D. Huber, and R. W. Schuckle, "System and method for providing performance in a personal gaming cloud," U.S. Patent 9 421 464, 2016.

[21] P. Parastar, F. Pakdaman, and M. R. Hashemi, "FRAME-SDN: A fair resource allocation for multiplayer edge-based cloud gaming in SDN," in *Proc. 25th ACM Workshop Packet Video*, New York, NY, USA, 2020, pp. 21–27.

[22] L. P. Qian, Y. Wu, H. Zhou, and X. Shen, "Joint uplink base station association and power control for small-cell networks with non-orthogonal multiple access," *IEEE Trans. Wireless Commun.*, vol. 16, no. 9, pp. 5567–5582, Sep. 2017.

[23] T. X. Tran and D. Pompili, "Joint task offloading and resource allocation for multi-server mobile-edge computing networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 1, pp. 856–868, Jan. 2019.

[24] H. Zhang, H. Liu, J. Cheng, and V. C. M. Leung, "Downlink energy efficiency of power allocation and wireless backhaul bandwidth allocation in heterogeneous small cell networks," *IEEE Trans. Commun.*, vol. 66, no. 4, pp. 1705–1716, Apr. 2018.

[25] C. You, Y. Zeng, R. Zhang, and K. Huang, "Asynchronous mobile-edge computation offloading: Energy-efficient resource management," *IEEE Trans. Wireless Commun.*, vol. 17, no. 11, pp. 7590–7605, Nov. 2018.

[26] H. Xiao, C. Xu, T. Cao, L. Zhong, and G.-M. Muntean, "GTTC: A low-expenditure IoT multi-task coordinated distributed computing framework with fog computing," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, 2019, pp. 1–6.

[27] F. Spinelli, A. Bazco-Nogueras, and V. Mancuso, "Edge gaming: A greening perspective," *Comput. Commun.*, vol. 192, pp. 89–105, Aug. 2022.

[28] T. Cao et al., "Reliable and efficient multimedia service optimization for edge computing-based 5G networks: Game theoretic approaches," *IEEE Trans. Netw. Service Manag.*, vol. 17, no. 3, pp. 1610–1625, Sep. 2020.

[29] N. McKeown et al., "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, 2008.

[30] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and H. Xie, "A survey on software-defined networking," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 1, pp. 27–51, 1st Quart., 2015.

[31] J. Du, C. Jiang, A. Benslimane, S. Guo, and Y. Ren, "SDN-based resource allocation in edge and cloud computing systems: An evolutionary Stackelberg differential game approach," *IEEE/ACM Trans. Netw.*, vol. 30, no. 4, pp. 1613–1628, Aug. 2022.

[32] M. Amiri, H. A. Osman, and S. Shirmohammadi, "Resource optimization through hierarchical SDN-enabled inter data center network for cloud gaming," in *Proc. 11th ACM Multimedia Syst. Conf.*, 2020, pp. 166–177.

[33] M. Min, L. Xiao, Y. Chen, P. Cheng, D. Wu, and W. Zhuang, "Learning-based computation offloading for IoT devices with energy harvesting," *IEEE Trans. Veh. Technol.*, vol. 68, no. 2, pp. 1930–1941, Feb. 2019.

[34] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji, and M. Bennis, "Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4005–4018, Jun. 2019.

[35] H. Hao, C. Xu, L. Zhong, and G.-M. Muntean, "A multi-update deep reinforcement learning algorithm for edge computing service offloading," in *Proc. 28th ACM Int. Conf. Multimedia*, 2020, pp. 3256–3264.

[36] Y. Zhang, J. Yao, and H. Guan, "Intelligent cloud resource management with deep reinforcement learning," *IEEE Cloud Comput.*, vol. 4, no. 6, pp. 60–69, Nov./Dec. 2017.

[37] T. Thein, M. M. Myo, S. Parvin, and A. Gawanmeh, "Reinforcement learning based methodology for energy-efficient resource allocation in cloud data centers," *J. King Saud Univ. Comput. Inf. Sci.*, vol. 32, no. 10, pp. 1127–1139, 2020.

[38] W. Zhang et al., "Deep reinforcement learning based resource management for DNN inference in industrial IoT," *IEEE Trans. Veh. Technol.*, vol. 70, no. 8, pp. 7605–7618, Aug. 2021.

[39] F. Bannour, S. Souihi, and A. Mellouk, "Distributed SDN control: Survey, taxonomy, and challenges," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 1, pp. 333–354, 1st Quart., 2018.

[40] K.-T. Chen, Y.-C. Chang, H.-J. Hsu, D.-Y. Chen, C.-Y. Huang, and C.-H. Hsu, "On the quality of service of cloud gaming systems," *IEEE Trans. Multimedia*, vol. 16, no. 2, pp. 480–495, Feb. 2014.

[41] T. Haarnoja et al., "Soft actor–critic algorithms and applications," 2018, *arXiv:1812.05905*.

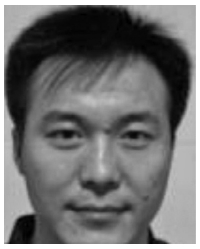
[42] T. P. Lillicrap et al., "Continuous control with deep reinforcement learning," 2015, *arXiv:1509.02971*.

[43] M. Han, Y. Lee, S. B. Moon, K. Jang, and D. Lee. "CRAWDAD dataset kaist/wibro (v. 2008-06-04)." 2008. [Online]. Available: https://crawdad.org/kaist/wibro/20080604

[44] X. Deng, J. Yin, P. Guan, N. N. Xiong, L. Zhang, and S. Mumtaz, "Intelligent delay-aware partial computing task offloading for multi-user industrial Internet of Things through edge computing," *IEEE Internet Things J.*, early access, Oct. 27, 2021, doi: 10.1109/JIOT.2021.3123406.

[45] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*.

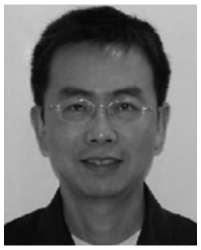
[46] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor–critic methods," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 1587–1596.



Xiaoheng Deng (Member, IEEE) received the Ph.D. degree in computer science from Central South University, Changsha, Hunan, China, in 2005.

Since 2006, he has been an Associate Professor and then a Full Professor with the Department of Electrical and Communication Engineering, Central South University, where he is a Joint Researcher with the Shenzhen Research Institute. His research interests include edge computing, Internet of Things, online social network analysis, data mining, and pattern recognization.

Prof. Deng is a Senior Member of CCF and a member of CCF Pervasive Computing Council and ACM. He was the Chair of CCF YOCSEF CHANGSHA from 2009 to 2010.



Honggang Zhang (Member, IEEE) received the B.S. degree from Central South University, Changsha, China, in 1991, the first M.S. degree from Tianjin University of China, Tianjin, China, in 1996, the second M.S. degree from Purdue University, West Lafayette, IN, USA, in 2000, and the Ph.D. degree in computer science from the University of Massachusetts at Amherst, Amherst, MA, USA, in 2006.

He is currently an Associate Professor of Computer Engineering with the Engineering Department, University of Massachusetts at Boston, Boston, MA, USA. His research interests span a wide range of topics in the area of computer networks and distributed systems. His current research focuses primarily on edge computing, mobile computing, and Internet of Things.

Dr. Zhang was a recipient of the National Science Foundation CAREER Award in 2009.

Jingjing Zhang received the B.Sc. degree in computer science and technology from Hunan Normal University, Changsha, China, in 2020. She is currently pursuing the Ph.D. degree in computer science and technology with the Wireless Communication and Mobile Computing Laboratory, Central South University, Changsha.

Her major research interests are edge computing, cloud gaming, and Internet of Things.

Ping Jiang received the master's degree in computer engineering from the University of Western Ontario, London, ON, Canada, in 2018. He is currently pursuing the Ph.D. degree with the Electrical and Communication Engineering Department, Central South University, Changsha, China.

His research interests include machine learning, natural language processing, computer vision, and security issues in neural networks.