future internet

MDPI

Article

Intelligent Video Streaming at Network Edge: An Attention-Based Multiagent Reinforcement Learning Solution

Xiangdong Tang, Fei Chen * and Yunlong He

College of Computer Science and Technology, Qingdao University, Qingdao 266071, China; 2020020680@qdu.edu.cn (X.T.); 2021020724@qdu.edu.cn (Y.H.)
* Correspondence: feic@qdu.edu.cn

Abstract: Video viewing is currently the primary form of entertainment for modern people due to the rapid development of mobile devices and 5G networks. The combination of pervasive edge devices and adaptive bitrate streaming technologies can lessen the effects of network changes, boosting user quality of experience (QoE). Even while edge servers can offer near-end services to local users, it is challenging to accommodate a high number of mobile users in a dynamic environment due to their restricted capacity to maximize user long-term QoE. We are motivated to integrate user allocation and bitrate adaptation into one optimization objective and propose a multiagent reinforcement learning method combined with an attention mechanism to solve the problem of multiedge servers cooperatively serving users. Through comparative experiments, we demonstrate the superiority of our proposed solution in various network configurations. To tackle the edge user allocation problem, we proposed a method called attention-based multiagent reinforcement learning (AMARL), which optimized the problem in two directions, i.e., maximizing the QoE of users and minimizing the number of leased edge servers. The performance of AMARL is proved by experiments.

Keywords: QoE; multiagent; reinforcement learning; edge computing

Citation: Tang, X.; Chen, F.; He, Y. Intelligent Video Streaming at Network Edge: An Attention-Based Multiagent Reinforcement Learning Solution. *Future Internet* **2023**, *15*, 234. https://doi.org/10.3390/fi15070234

Academic Editor: Guan Gui

Received: 3 June 2023
Revised: 23 June 2023
Accepted: 27 June 2023
Published: 3 July 2023

1. Introduction

The content delivery networks (CDNs) deployed on the edge server are committed to satisfy as many user service requests as possible and improve user experience, especially high-quality video services. Nowadays, due to the improvement in video resolution and the increase in real-time interaction requirements, it is unrealistic to only upgrade the backbone network. Therefore, new challenges are brought to the edge network servers that provide nearby services and a video streaming system. Typically, the capacity or the service area of each edge server is limited, and how to assign users to different servers is called the EUA (edge user allocation) problem, which has been widely explored in recent years [1]. It is modeled as a boxing problem, and an optimal approach is proposed based on dictionary goal programming techniques [1], and adaptive services are provided for mobile users with heterogeneous needs [2]. Furthermore, mobile video streaming typically occurs in a nonstationary environment, including user movement, intermittent connection, user onboarding, or leaving. Hence, it is a challenge to obtain an optimal bitrate in a real-world implementation, especially when solving the problem of user allocation in overlapping regions [3].

In large-scale video service systems, there are methods that use single-agent reinforcement learning to solve the above two problems independently [4,5]. On the one hand, the bitrate selection is highly dependent on the network connection between the edge server and the user, and the service resource allocation is affected by the users' quality of experience (QoE). On the other hand, dynamic factors, such as the number, behavior, and network status of users, lead to sudden changes in the optimal solution. Consequently, it is inappropriate to address the two issues independently. Although the approaches taken

by these two solutions to server assignment and bitrate adaption vary, they both strive to improve user QoE while utilizing the least amount of infrastructure resources or system costs. These two issues are typically framed as combinatorial optimization problems and resolved separately one after the other. However, these two issues are strongly interrelated, and we are motivated to propose an end-to-end approach based on multiagent reinforcement learning (MARL) that integrates the above two problems, which treats each edge server as an agent for policy execution and improves the policy through feedback from users' experience, thereby continuously improving the quality of service of all edge servers. To better realize the cooperative relationship between edge servers, we achieve information sharing among edge servers by introducing an attention mechanism, then obtaining the final service policy through centralized training and decentralized execution.

The main contributions of this article are summarized as follows:

- We model the problems of user allocation and bitrate selection as a decentralized partially observable Markov decision process (Dec-POMDP).
- We propose an end-to-end approach based on multiagent reinforcement learning that can simultaneously consider the edge user allocation and bitrate selection problems.
- We use the attention mechanism to achieve the information sharing among edge servers, then obtain the final service policy through centralized training and decentralized execution.

Therefore, an attention-based MARL (AMARL) is designed to tackle the EUA problem and bitrate selection simultaneously under a dynamic environment. The remainder of the article is organized as follows: The literature is explored in Section 2. The details of the method are described in Section 3. The experiments are implemented in Section 4. Finally, we conclude this paper in Section 5.

2. Related Work

In this section, we will introduce the related work from three aspects, namely, edge user allocation, adaptive bitrate, and multiagent reinforcement learning. An overview of related works is shown in Table 1.

2.1. Edge User Allocation

With the rapid increase in data traffic, CDN has been widely studied [6,7]. For instance, in [8], the authors proposed an edge cache-based intelligent content delivery solution to improve network performance in information-centric wireless networks. Edge user allocation (EUA) was usually considered as one of the most critical problems in sophisticated edge environments. The study was divided into two categories—static user assignment and dynamic user assignment—according to whether the user's location was moving or not. In static user assignment, for example, this problem was modeled as a variable-size vector boxing problem for utility efficiency [1] or QoE improvement [2]. The authors proposed a decentralized game-theoretic method to select each user's channel and edge server while meeting their resource and data rate requirements [9]. In dynamic user assignment, this problem was modeled as an online decision and evolutionary process and proposed a mobile-aware and migration-supporting approach, called MobMig, for assigning users in real time [10]. An online approach called OL-EUA was proposed for solving dynamic EUA problems in NOMA-based MEC systems [11]. In [12], the authors proposed a centralized training and distributed execution multiagent dueling double deep Q network solution to maximize the total network data rate while minimizing the mobility-induced handoffs. However, none of these methods considered the optimization of the long-term QoE, while assigning users did not select the appropriate video bitrate to the users at the same time.

2.2. Adaptive Bitrate

To enhance the user's QoE, the adaptive bitrate algorithms should be taken into account, which can dynamically select different video bitrates for the next video segment

playback according to either the network status [13,14] or buffer conditions [15,16]. For the network state-based methods, Liu et al. proposed a novel rate-adaptive algorithm for adaptive HTTP streaming based on segment fetch time (SFT) to detect bandwidth changes [17], and further proposed the ratio of expected SFT as a new rate-adaptive metric to quickly detect network congestion and idle network capacity [18]. For the buffer-based methods, Huang et al. proposed an approach that began by using only the buffer, and then asked when capacity estimation was needed [15]. BOLA was devised by using Lyapunov optimization techniques to minimize rebuffering and maximize video quality [16]. However, both of the above two methods are highly dependent on the assumption of prediction accuracy, which can be hardly competent in the sophisticated environment and heterogeneous user demand. In [19], the authors proposed a method based on a reinforcement learning algorithm to select the bitrates of the region of interest adaptively for panoramic videos. In [20], in order to optimize the QoE, the authors proposed a novel ABR algorithm considering the user preference based on short trajectory segments.

Table 1. Overview of related works.

Papers	Description	Edge User Allocation	Adaptive Bitrate	Reinforcement Learning
[17]	Detecting bandwidth changes using smoothed HTTP throughput based on the segment fetch time		✓	
[18]	Investigating rate adaptation for the serial segment fetching method and the parallel segment fetching method in a content distribution network		✓	
[14]	Developing a set of techniques to trade off stability, fairness, and efficiency in the video adaptive framework		✓	
[13]	Designing a video bitrate adaptive algorithm at the application layer		✓	
[15]	Proposing the video buffer to ease the need for capacity		✓	
[1]	Modeling the edge user allocation problem as a bin packing problem	✓		
[2]	Enabling flexible levels of QoE for app users	✓		
[19]	Selecting the bitrates of the region of interest adaptively for panoramic videos		✓	✓
[10]	Considering the edge user allocation problem as an online decision-making and evolvable process	✓		
[20]	Considering user preference based on short trajectory segments		✓	✓
[16]	Using Lyapunov optimization to minimize rebuffering and maximize video quality		✓	
[9]	Investigating the EUA problem in a NOMA-based MEC system	✓		
[11]	Solving the dynamic EUA problem with mutual interference between users	✓		
[5]	Predicting resource utilization of user requests for better user allocation	✓		✓
[12]	Maximizing the data rate of the total network and prioritizing the quality of service for key users	✓		✓

2.3. Multiagent Deep Reinforcement Learning

Deep reinforcement learning overcomes the limitations of reinforcement learning and uses deep neural networks to approximate value functions, effectively improving the learning speed and performance of reinforcement learning [21]. A reinforcement learning algorithm has been widely used [22,23]. To address the worse performance in the multiagent environment, QMIX adopts the framework of centralized learning and distributed execution, but the characteristic limits its applicability to cooperative scenarios [24]. As a further improvement of the QMIX, QTRAN transforms the original joint action-value

function into an easily factorizable one, with the same optimal actions [25]. MADDPG takes into account the action policies of other agents and is able to successfully learn policies that require complex multiagent coordination [26]. It uses centralized training and decentralized execution to achieve remarkable results and can be applied to collaborative, competitive, and mixed environments.

3. Problem Formulation and Model Design

In this section, we will introduce the problem formulation and the system model design, and then analyze the computational complexity of our proposed algorithm. Table 2 lists the symbols used in this paper.

Table 2. Notation.

Notation	Description
$E = \{e_1, e_2, \cdots, e_i\}$	Finite set of edge servers e_i, where $i = 1, 2, \cdots, m$
$U_i = \{u_1, u_2, \cdots, u_j\}$	Finite set of users u_j under the coverage of the edge server e_i
C_i	The set of users set served by the edge server e_i, $C_i \subseteq U_i$
Con_i	The maximum number of connections to the edge server e_i
d_{ij}	Geographic distance between the edge server e_i and user u_j
$rtt_i(u_j)$	The RTT value of the user u_j with the corresponding server e_i
$th_i(u_j)$	Throughput rate of the user u_j under the coverage of the edge server e_i
$b_i(u_j)$	The video resolution assigned to the user u_j
$lb_i(u_j)$	The last video resolution assigned to the user u_j
$ap(\lvert e_i \rvert)$	Decide whether to penalize the reward based on the number of user sets assigned to the edge server e_i
$tp(th_i(u_j), b_i(u_j))$	Decide whether to penalize the reward based on the $th_i(u_j)$ and $b_i(u_j)$ of the user u_j under the coverage of the edge server e_i
$QoE(u_j)$	QoE of the user u_j
EN	Number of leased servers
S_{ij}	State information between the server e_i and user u_j
S_i	State vector of the server e_i, consisting of the status $S_{ij}, j \in U_i$ and $\lvert C_i \rvert$
$\mathcal{S}$	A state matrix of all server states
A_{ij}	The action of the server e_i to the user u_j, which is a 2-tuple where the first element is the server index and the second element is the assigned resolution $b_i(u_j)$
A_i	Action vector between the server e_i and its covered users U_i
R_i	Reward for the edge server e_i

3.1. Problem Formulation

As shown in Figure 1, there are distributed edge servers and mobile users in the edge network video streaming system. Each end user has different trajectories under specific modes of transportation, so the network connection status with adjacent edge nodes is constantly changing. As mentioned earlier, every edge server with limited resources needs to adapt to various requests online in real time, which is called the EUA problem. At the same time, each mobile user also needs to choose an appropriate bitrate based on their local network connection quality to ensure smooth playback, which is known as the bitrate selection problem. Although these two problems aim to achieve different strategies, such as server allocation or bitrate selection, their goals are similar, namely, to maximize users' QoE with limited infrastructure resources or minimizing system costs.

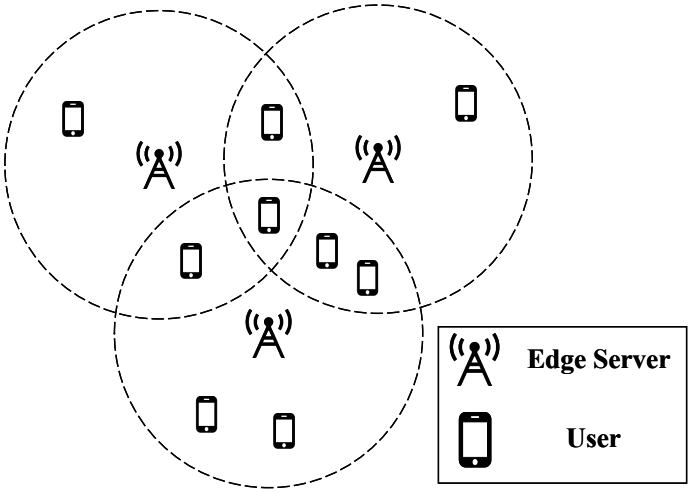

Figure 1. System overview.

Given a video streaming system with distributed edge servers and mobile users, their interaction process is shown in Figure 2 under the AMARL framework. Let us denote the set of edge servers as $E = \{e_1, e_2, \cdots, e_i\}$, the users set under the coverage of e_i as $U_i = \{u_1, u_2, \cdots, u_j\}$, the users set served by e_i as $C_i (C_i \subset U_i)$, and the maximum number of connections to e_i as Con_i. Denote geographic distance, round-trip time (RTT), and throughput rate between e_i and u_j as d_{ij}, $rtt_i(u_j)$, and $th_i(u_j)$, respectively. $b_i(u_j)$ and $lb_i(u_j)$ represent the current and last video bitrate assigned to u_j. Then the users' QoE can be defined as

$$QoE(u_j) = bw \times b_i(u_j) - sw \times |b_i(u_j) - lb_i(u_j)| + tw \times tp(th_i(u_j), b_i(u_j)) - rw \times rtt_i(u_j). \quad (1)$$

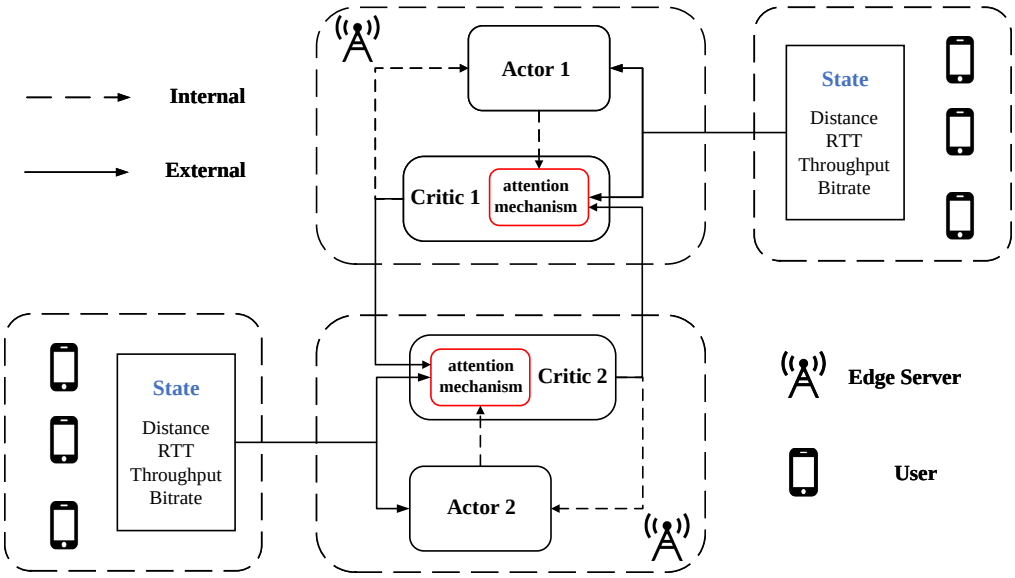

Figure 2. An overview of the system structure.

Note that the absolute value of $b_i(u_j) - lb_i(u_j)$ is used to reflect the smoothness of the video bitrate change. Intuitively, if this value is too large, the user will feel that the video playback is discontinuous. In addition, it is necessary to ensure that the throughput $th_i(u_j)$ between u_j and e_i can support the selected bitrate $b_i(u_j)$, so the throughput penalty function $tp(\cdot)$ is defined as

$$tp(th_i(u_j), b_i(u_j)) = \begin{cases} th_i(u_j) - b_i(u_j), & th_i(u_j) < b_i(u_j) \\ b_i(u_j), & th_i(u_j) \geq b_i(u_j) \end{cases} \tag{2}$$

Therefore, bw, sw, tw, and rw are used as the weights of bitrate, smoothness, throughput, and RTT, respectively, in (1).

3.2. Model Design

A fully cooperative multiagent task can be described as a Dec-POMDP, which can be denote as a tuple $G = \langle S, A, P, r, Z, O, n, \gamma \rangle$. $s \in S$ denotes the true state of the environment. At each step, each agent $m \in \mathcal{M} = \{1, ..., M\}$ chooses an action $a_m \in A$, and all actions form joint action $\boldsymbol{a} \in \boldsymbol{A} = A^M$. P is the state transition function depending on the environment. When all agents complete the decision, the environment will transit from the state s to the next state s' according to the state transition function $P(s'|s, \boldsymbol{a}) : S \times \boldsymbol{A} \times S \to [0, 1]$, and return the reward at the same time. All agents share the same reward function $r(s, \boldsymbol{a}) : S \times \boldsymbol{A} \to \mathbb{R}$. $\gamma \in [0, 1)$ is the discount factor. For a Dec-POMDP, the environment is partially observable for agents. Each agent obtains its individual observation $z \in Z$ according to observation function $O(s, m) : S \times M \to Z$. Each agent has an action-observation history $\tau_m = (a_{m,0}, z_{m,1}, ..., a_{m,t-1}, z_{m,t})$, on which it conditions a stochastic policy $\pi_m(a_m|\tau_m) \in [0, 1]$. The joint policy $\boldsymbol{\pi}$ has a joint action-value function $Q_{\boldsymbol{\pi}}(s_t, \boldsymbol{a}_t) = \mathbb{E}_{s_{t+1:\infty}, \boldsymbol{a}_{t+1:\infty}}[R_t|s_t, \boldsymbol{a}_t]$, where $R_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$ is the discounted return. The goal of agents is to find a joint policy $\boldsymbol{\pi}$ to maximize the expected discounted reward. In MARL, agents learn to make decisions by exploring the unknown environment and using the feedback received from the environment. In this setting, the objective of each agent is to maximize the shared reward.

Then the essential elements of a Dec-POMDP can be defined as follows:

State. In this scenario, due to the difference in factors, such as latency between each user and each edge server, we define the state of a user u_j under the coverage of a single edge server e_i as S_{ij}, and each S_{ij} is a 4-tuple as shown in (3).

$$S_{ij} = (d_{ij}, rtt_i(u_j), th_i(u_j), b_i(u_j)) \tag{3}$$

Then the input state of the edge server e_i is

$$S_i = (S_{i1}, S_{i2}, \cdots, S_{i|U_i|}, |C_i|) \tag{4}$$

In (4), we input not only the states of all users under the coverage of the edge server e_i into the neural network but also the number of remaining connections to e_i. In AMARL, the input to the actor network is the respective state S_i of each agent, while in the critic network, since the training is to be centralized, the input is the set of states $\mathcal{S}$ of each edge server e_i, defined as shown in (5):

$$\mathcal{S} = \begin{bmatrix} S_{11} & S_{12} & \cdots & S_{1|U_1|} & |C_1| \\ S_{21} & S_{22} & \cdots & S_{2|U_2|} & |C_2| \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ S_{m1} & S_{m2} & \cdots & S_{m|U_m|} & |C_m| \end{bmatrix} \tag{5}$$

Because of the presence of other agents, the prediction of a single actor is not a global joint action. In order to reduce the impact of other agents on the current critic's evaluation of the actor's prediction, a masked array is used to eliminate useless information in the evaluation. The algorithm for state matrix construction is shown in Algorithm 1.

Algorithm 1: State matrix construction algorithm.

Input: Edge server set E
Output: Current state $\mathcal{S}$
for $e_i \in E$ **do in parallel**
  Collect the all information of the user $u_j \in U_i$ that is covered by the edge server e_i;
  for $u_j \in U_i$ **do in parallel**
    Return the geographical distance d_{ij} from the server e_i, communication delay $rtt_i(u_j)$, throughput $th_i(u_j)$, current video resolution $b_i(u_j)$;
  end
  Construct the state S_{ij} of the current edge server e_i and each user u_j according to (3);
  Calculate the number of users served by the edge server e_i, $|C_i|$;
  Construct the current state vector according to (4);
end
Obtain the current state $\mathcal{S}$ of all servers according to (5);

Action. For each agent, the action is initially set to a list of binary groups, where each element of the list is a binary group made up of the user index and the assigned video bitrate. Denote the action of e_i for u_j as A_{ij}. Further, A_{ij} is a binary as shown in (6).

$$A_{ij} = (i, b_i(u_j)) \tag{6}$$

Thus, the action vector A_i predicted by the edge server e_i is

$$A_i = (A_{i1}, A_{i2}, \cdots, A_{i|U_i|}) \tag{7}$$

However, it is a challenge to determine the actions because each agent will have access to a list of all user behaviors, requiring several decisions to be made about each user by various agents, which will certainly result in some conflict. Section 3.3 describes the detailed solution.

Reward. In the EUA problem, the goal is to maximize the number of assigned users and minimize the number of leased servers. However, if only the user's QoE and the number of leased servers are considered as the reward function, the capacity of each agent will be ignored. Therefore, a penalty function is introduced to prevent the number of server connections from exceeding their own capacity. Denote R_i as the reward of edge server e_i and EN as the number of leased servers; then

$$R_i = \sum_{u_j \in C_i} QoE(u_j) + ap(e_i) - EN \tag{8}$$

Furthermore, in order to serve as many users as feasible within the coverage area, the penalty function $ap(\cdot)$ is defined in accordance with the number of connections and the number of users inside the coverage area as follows:

$$ap(e_i) = \begin{cases} Con_i - |C_i|, & |C_i| > Con_i \\ |C_i|, & |C_i| \leq Con_i \end{cases} \tag{9}$$

Actor. For each UAV agent, there is an actor and a critic. The actor is the network parameterized policy function $\mu(s_i^j; \theta^\mu)$; θ^μ is the parameter of the network. The action of the agent can be obtained by the deterministic policy $a_t = \mu(s_i^j; \theta^\mu)$. After all agents

complete the action selection, the joint action a_t can be obtained. The gradient can be calculated as

$$\nabla_{\theta_i} J = \frac{1}{S} \sum_j \nabla_{\theta_i} \mu_i(s_i^j) \nabla_{a_i} Q_i^{\psi}(\mathcal{S}^j, a_1^j, \cdots, a_i, \cdots, a_N^j)|_{a_i = \mu_i(s_i^j)} \tag{10}$$

where i is the agent index, S is the batch size, μ is the actor policy, and Q is the critic evaluation.

Critic. In order to more accurately evaluate the action, each agent queries other agents for information about its observations and actions and integrates it into the estimate of its value function. To compute the Q-value function $Q_i^{\psi}(s,a)$ for $agent_i$, the critic receives the states $s = (s_1, \ldots, s_N)$ and actions $a = (a_1, \ldots, a_N); i \in 1...N$. $Q_i^{\psi}(s,a)$ is a function of states and actions of $agent_i$ and the contributions of other agents:

$$Q_i^{\psi}(s,a) = f_i(g_i(s_i, a_i), x_i) \tag{11}$$

where f_i is a two-layer MLP, while g_i is a one-layer MLP embedding function. Denote the contribution from other agents as x_i, which is a weighted sum of each agent's value:

$$x_i = \sum_{j \neq i} \alpha_j v_j \tag{12}$$

where the value, v_j is the embedding of $agent_j$. The attention weight α_j considers the correlation of $agent_i$ and $agent_j$; that is, it is calculated according to the distance between the two agents and the overlap rate of users in the service area. The details are shown in Figure 3.

As shown in Figure 2, the critics of each agent can obtain the state and action information of other agents extracted by the attention mechanism during training for an objective action evaluation. After evaluation, each agent independently predicts the next action based on the evaluation value.

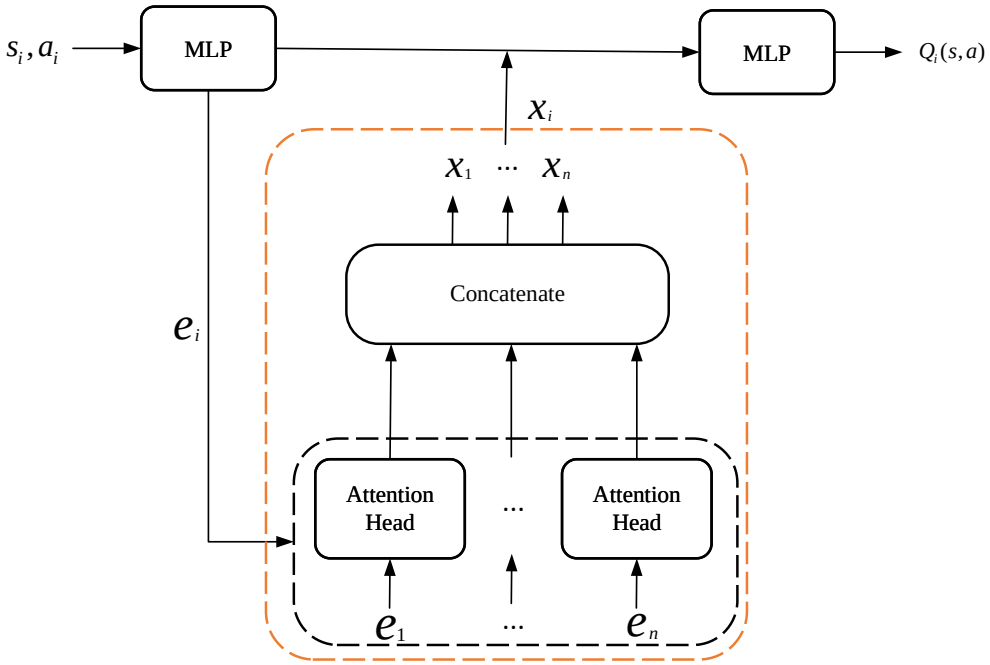

Figure 3. Calculating $Q_i^{\psi}(s,a)$ for $agent_i$. Each agent encodes its observations and actions, sends them to the central attention mechanism, and receives a weighted sum of other agents' encodings.

3.3. Perform EUA and Bitrate Selection Simultaneously

As shown in Figure 4, in the middle of each time period, each agent acquires the users receiving data within its own coverage area and obtains its state S_i. The input states S_i of each agent are combined as the current set of states $\mathcal{S}$, which is fed into each agent's actor network to obtain the predicted action A_i. To solve the action conflict problem between different agents and optimize the user's QoE, the action that brings the best QoE to the user will be executed, after calculating the QoE under different actions. In this way, the set of actions A and the set of rewards R for all agents are obtained, and the next state $\mathcal{S}'$ is influenced by the current action and the next time period of user information. Additionally, when an epoch ends, $done = Ture$ is set and the rest of the time is set to $False$. The 5-tuple $[\mathcal{S}, A, R, \mathcal{S}', done]$ is stored in the experience pool for training. When there are enough training data, training starts. The training process follows the principle of centralized training and step-by-step execution; i.e., the actor predicts the data individually for each agent with its own observed data, while the critic trains by taking the state $\mathcal{S}$ and action set A of all agents as input to estimate the value. After training, we just need to test each agent's actor with the test set. The specific process is shown in Algorithm 2.

Algorithm 2: AMARL.

Input: Initialized model parameters Θ, edge server set E, the maximum value M of the episode
Output: Actor network model for edge servers

Initialize the experience pool ReplayBuffer.
for *episode = 1 to M* **do**
  for *t=1 to T* **do**
    Execute Algorithm 1 to obtain the current state $\mathcal{S}$;
    Each agent obtains the action $A_i = \mu(s_i^j; \theta^\mu) + \mathcal{N}_t$
    Each edge server e_i executes the action A_i according to the current strategy to obtain a new state $\mathcal{S}'$, and calculates the reward R_i according to (8);
    if $\mathcal{S}' = \mathcal{S}$ **then**
      done = True
    else
      done = False
    end
    Store $[\mathcal{S}, A, R, \mathcal{S}', done]$ into ReplayBuffer.
    $\mathcal{S} \leftarrow \mathcal{S}'$;
    for $e_i \in E$ **do in parallel**
      Sample data sample j from ReplayBuffer;
      Calculate the estimated value $y_i^j = R_i + \gamma Q_i^\psi(\mathcal{S}, A)$ according to (11);
      Based on this estimate, the parameters of the critic network are updated by minimizing the loss value;
      Update the parameters of the actor network Θ_i according to the loss value and (10).
    end
  end
end

3.4. Computational Complexity Analysis

In this part, we analyzed the computation complexity of our proposed algorithm. During the training phase, the computational complexity of the single agent algorithm is mainly related to the number of layers of the deep neural network (DNN) used by each agent and the number of neurons used in each layer [27]. In addition, our algorithm is a multiagent algorithm; it is also necessary to consider the impact of the number of agents.

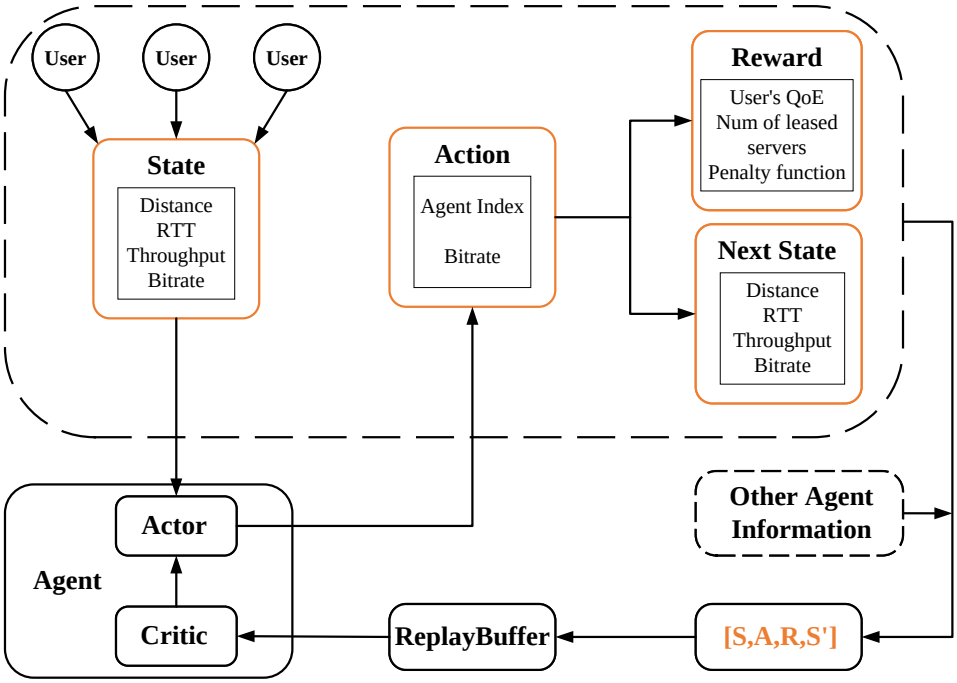

Figure 4. Process of the training.

In the training phase, the computational complexity for a single DNN to both forward propagation and update in a single step is $\mathcal{O}(S(\sum_{i=1}^{L} n_i n_{i+1}))$, where S is the minibatch size, L represents the number of layers, and n_i denotes the number of neurons in the i-th layer. Let F_a, F_c represent the computational complexity of the actor and critic of a single agent, respectively. F_a, F_c can be calculated as

$$F_a = \mathcal{O}(TMS(\sum_{i=1}^{L_a} n_i n_{i+1})), \quad F_c = \mathcal{O}(TMS(\sum_{i=1}^{L_c} n_i n_{i+1})) \tag{13}$$

where T represents the number of steps in each episode, M represents the total number of episodes, and L_a and L_c represent the number of layers of actor and critic, respectively. Let N represent the number of agents. Then the computational complexity of our algorithm in the training phase can be expressed as

$$C_{train} = \sum_{1}^{N}(F_a + F_c) \tag{14}$$

During the distributed execution phase, each agent only needs to use a trained actor network to select action, so the computational complexity of our algorithm is

$$C_{execute} = \sum_{1}^{N}(\mathcal{O}(\sum_{i=1}^{L_a} n_i n_{i+1})) \tag{15}$$

4. Evaluation

In this section, we analyzed the influence of hyperparameters on convergence and evaluated the algorithm performance.

4.1. Settings

Simple experiment. We will conduct experiments in a simulated environment; the experimental data use the real data of the 4G/LTE dataset. We compare the adaptive performance of our model with a simple experiment where throughput drops suddenly.

On the server side, the system is Ubuntu 16.04 with the Apache HTTP server version 2.2.22. On the client side, the system uses Windows 10, and the CPU is Intel(R) Core(TM) i5-8400. For live connections, the server's timeout is set to 100s, and the video stored on the server is subdivided into simultaneous 2 s video clips. Additionally, there are six available bitrate versions for each clip. They are 300, 750, 1200, 1850, 2850, and 4300 kbps, respectively. The initial buffer delay is equal to 10 durations (20 s).

Complex experiment. The complex environment has three edge servers, and the penalty function for the number of connections is distributed at set coordinates, from which the data of each user are calculated. We designed three allocation strategies to compare with our model:

- Greedy strategy. At each time slot, each user is assigned to the closest agent as long as there are remaining connections on that edge server, and the streaming bitrate transmitted is based on the Pensieve model.
- Random strategy. Each user is randomly assigned to an edge server, as long as there are remaining connections on that edge server, and the streaming bitrate transmitted is based on the Pensieve model.
- Step-by-step reinforcement learning. Reinforcement learning is used to implement user assignment, and then a Pensieve model is used to select the bitrate of the video stream.

Implementation Details. We have set the learning rate $\alpha = 0.0002$, the discount factor $\gamma = 0.9$, the experience pool size buffer_size $= 10{,}000$, and the batch size per training batch_size $= 16$. For the calculation of QoE in (1), we set bw, sw, tw, and rw to be 1, 1, 1.5, and 0.01, respectively. Based on the Ornstein–Uhlenbeck process [28], a noise $\mathcal{N}$ is introduced in order to add a bit of randomness and increase the learning coverage when selecting actions. It starts with an exploration factor $\epsilon = 1$ that decreases to a minimum of 0.01 at the end of the training, with a decay factor of 0.999, indicating that the noise has less and less influence. Although it will increase the training time, the soft update factor $\tau = 0.01$, which will increase the stability of the training. The parameters are shown in Table 3.

Table 3. Parameter values.

Parameter	Value
Replay buffer capacity	1×10^5
Optimizer	Adam
Learning rate α	2×10^{-4}
Discount factor γ	0.9
Minibatch size N	16
Soft update constant ϵ	0.01
Number of training episodes	1000

4.2. Convergence Analysis

In this part, we analyzed the impact of different hyperparameter settings on convergence performance to evaluate the convergence performance of our proposed algorithm.

We first evaluated the impact of different learning rate settings on convergence performance. We evaluated the convergence performance under three settings, where the learning rate is set to 2×10^{-5}, 2×10^{-4}, and 2×10^{-3}, respectively. The result is shown in Figure 5. It can be seen that in the three learning rate settings, the rewards obtained by the agent increases with the training, and can eventually converge. When the learning rate is 2×10^{-4}, it has the best convergence performance. Too small or too large learning rate will lead to worse convergence performance.

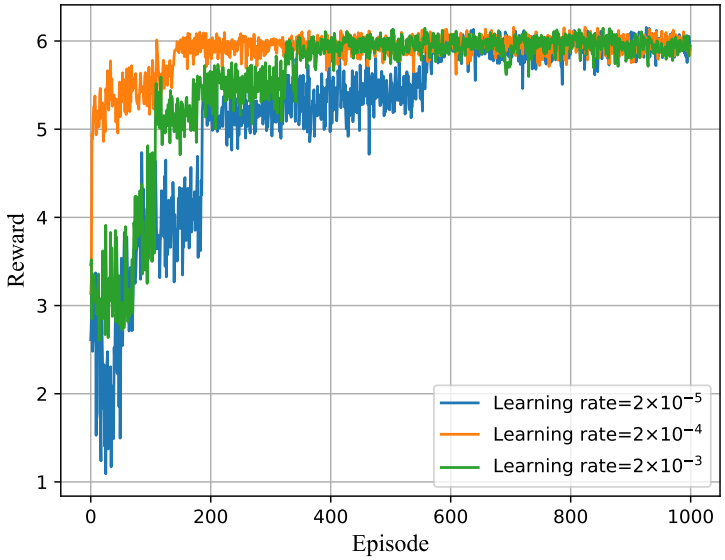

Figure 5. Comparison of different learning rates.

The convergence performance of the algorithm is also affected by the minibatch size. The minibatch size determines the number of samples used in gradient updates, which directly affects the learning speed and stability of the algorithm. We compared the convergence performance of our algorithm with minibatch sizes of 8, 16, and 32. The result is shown in Figure 6. The algorithm can converge under all three minibatch sizes. Compared with other settings, the algorithm can converge faster when the minibatch size is 16. When the minibatch size is set to 8, the convergence speed of the algorithm is slow. Because the number of samples is too small during a single update, the learning speed decreases.

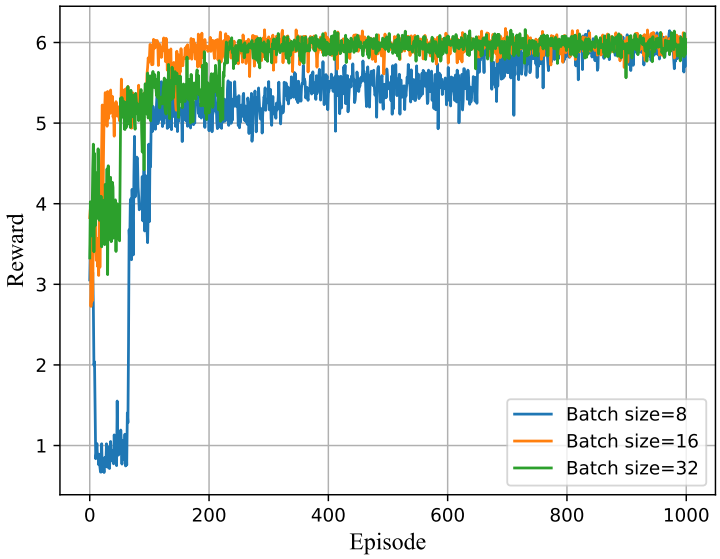

Figure 6. Comparison of different minibatch sizes.

The discount factor also affects the convergence performance of the algorithm. Figure 7 shows the convergence performance comparison of the algorithm under different discount factor settings. It can be seen that the algorithm has the best convergence performance when the discount factor is set to 0.9. A larger discount factor indicates that the agent places more emphasis on long-term benefits. When the discount factor is set to 0.7 and 0.8, the

agent's emphasis on long-term benefits decreases and becomes more shortsighted, thus affecting convergence performance.

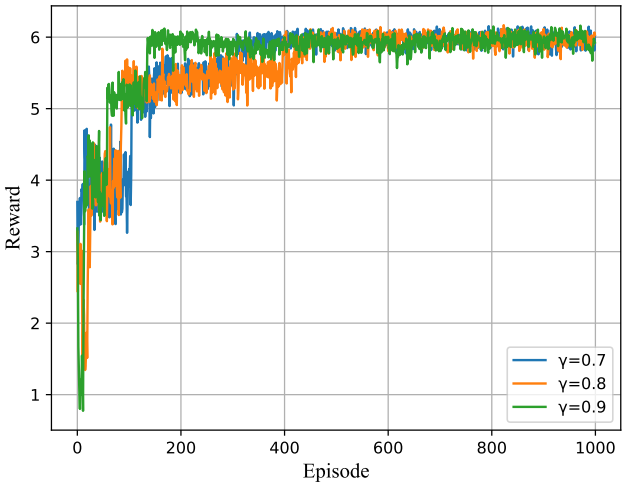

Figure 7. Comparison of different discount factors.

4.3. Results of the Simple Experiment

Figure 8 shows that the throughput suddenly dropped from 4500 to 350 kbps at 30 s and gives bitrates and caching behavior for all methods. When throughput drops, a rate-based algorithm responds first and quickly drops to 350 kbps, so the buffer pool does not consume much. Although AMARL responded quickly, the drop was smoother compared with the rate-based algorithms and increased the smoothness of playback. However, the buffer-based algorithm did not respond at the beginning. When the buffer pool drops to the threshold, the buffer-based algorithm starts to adjust the video bitrate. The rate-based method can reflect network changes rapidly, the changes are more drastic, and the smoothness is very low compared with the proposed model. In contrast, buffer-based methods try to keep the video bitrate high, but the buffer level drops quickly. Then the client has to switch to the lowest video bitrate. Unlike other methods, our method considers both the video bitrate and the smoothness of the change of the bitrate and has little influence on the user experience.

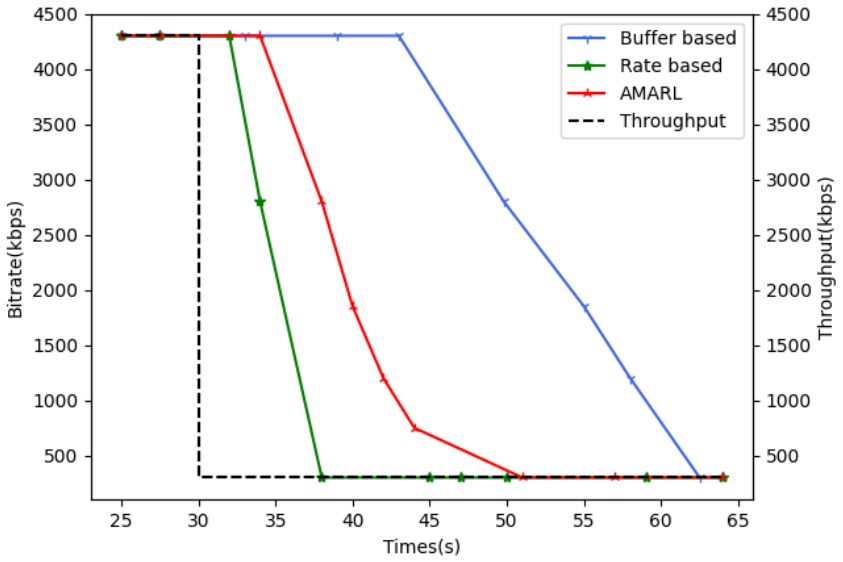

Figure 8. Variation of the bitrate of different algorithms when the throughput drops suddenly.

4.4. Dynamic Environment Complex Experiment

Given the dynamic user behavior and heterogeneous network conditions, we know that multiagent solutions can achieve relatively stable results and stay near these results for a long time. We compare the performance of multiagent deep deterministic policy gradient (MADDPG) and AMARL models with added attention mechanisms in more complex situations. MADDPG is an algorithm designed for multiagent, continuous behavior spaces. The predecessor of MADDPG is DDPG. The purpose of the DDPG algorithm is to solve the problem of reinforcement learning in the continuous behavior space. MADDPG is improved on the basis of DDPG to make it suitable for cooperative task learning between multiagents. The results are shown in Figure 9. The agents trained under AMARL can effectively use all the information received, and can better adapt to complex and changeable dynamic environments.

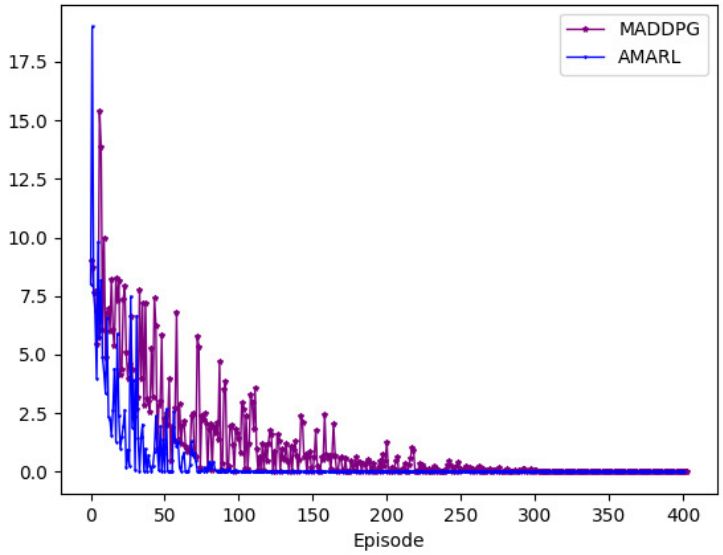

Figure 9. Convergence speed of AMARL and MADDPG.

Further, we also compare the average QoE and average bitrate of different methods. Figure 10 shows the average QoE of users and the average bitrate of video streams for four different methods. As can be seen from the figure, both AMARL and step-by-step reinforcement learning can make the average bitrate of the video stream reach a higher value, but the user QoE of AMARL is 70% higher than it. For both greedy and random strategies, both indicators of AMARL are well above their values.

We also explored the performance under different numbers of users and edge servers. Figures 11 and 12 show the trend of each parameter depending on the number of users or agents. Figures 11a and 12a demonstrate that the model obtained by AMARL has better performance in average user QoE compared with other methods. For the bitrate allocation shown in Figures 11b and 12b, the effect of AMARL is weaker than that of the Step-by-step reinforcement learning method, but in terms of the smoothness shown in Figures 11c and 12c, the effect of AMARL is significantly better than that of the step-by-step reinforcement learning method. This also reflects the importance of the adaptive bitrate algorithm. It should be noted that from (1), the smaller the smoothness, the better the user experience. In summary, our proposed model performs well in complex dynamic environments and always maintains a relatively high bitrate. Moreover, the change in bitrate is relatively smooth. Through experiments on Pensieve, it can be seen that although the same parameters and dataset are used, the training results vary greatly, probably because the input of Pensieve contains RTT, and the users selected by the greedy and random strategies are different before each training; i.e., the selection of RTT is also different. Table 4 shows a comparison of the number of servers used by different algorithms under

different numbers of agents. It can be seen that our proposed algorithm uses the fewest number of servers under different numbers of agents.

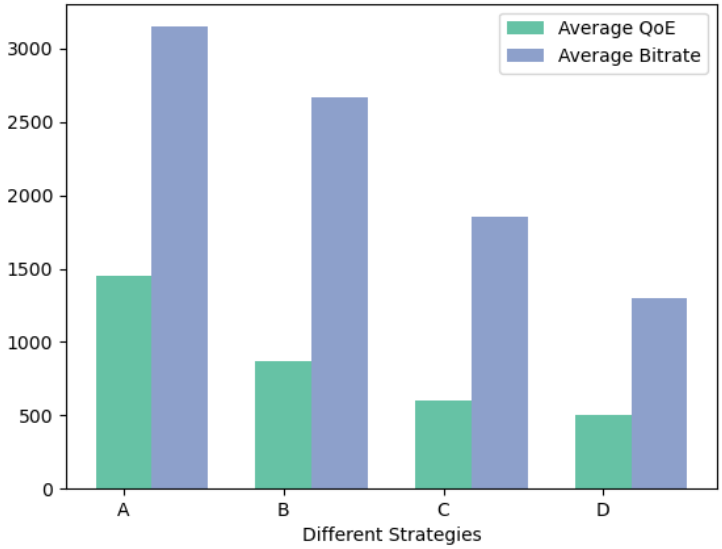

Figure 10. Experimental results of (**A**) AMARL, (**B**) step-by-step reinforcement learning, (**C**) greedy strategy, and (**D**) random strategy.

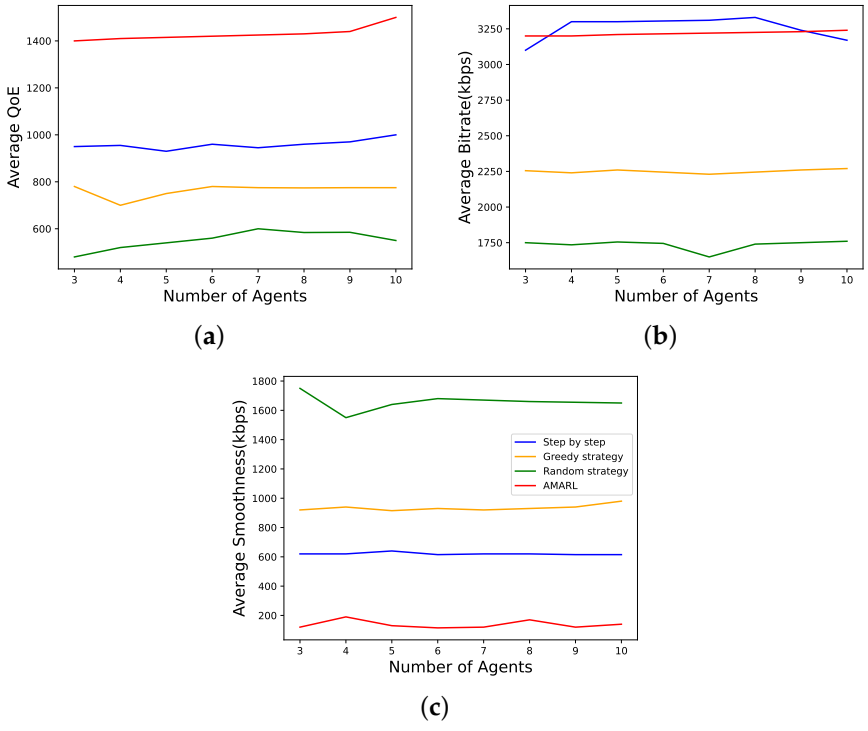

Figure 11. Parameter variation trend as the number of agents changes: (**a**) QoE, (**b**) bitrate, and (**c**) smoothness.

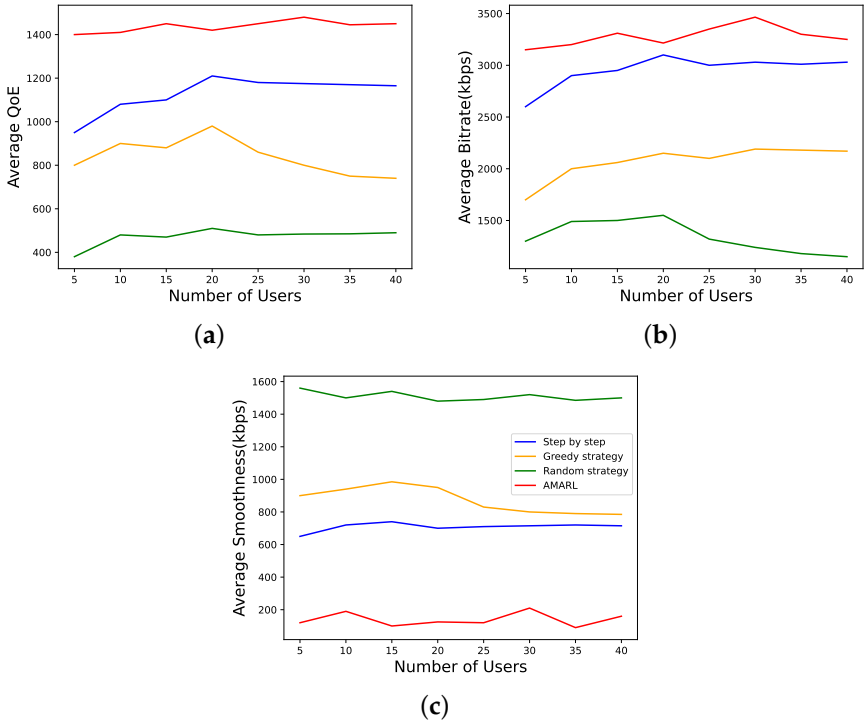

Figure 12. Parameter variation trend as the number of users changes: (**a**) QoE, (**b**) bitrate, and (**c**) smoothness.

Table 4. Number of servers used.

Number of Agents	4	5	6	7	8	9
AMARL	3	3	4	4	5	5
Greedy strategy	3	4	5	6	6	7
Random strategy	4	5	6	7	7	8
Step by step	3	4	4	5	5	6

5. Conclusions

To tackle the EUA problem in a variety of environments, we propose an attention-based multiagent reinforcement learning (AMARL) algorithm, which can well adapt to the dynamic environment and learn in the unknown environment, and jointly consider user allocation and bitrate selection in the reward function, to achieve the long-term QoE guarantee of users, and we applied the attention mechanism to the critic to reflect the influence of different agents. Then, we conducted experiments through scenario simulations, and our experiments showed that our approach significantly outperformed the Pensieve-based random and greedy policies. The algorithm could analyze the optimal allocation scheme for edge servers in real time in complex and variable environments to maximize the QoE for users. Our future direction is to realistically simulate the process of video transmission from the edge server and to specify the edge server resources beyond the simple number of connections. At the same time, we would also consider more influencing factors, such as security and network dropouts.

Author Contributions: Conceptualization, X.T. and F.C.; methodology, X.T.; validation, Y.H.; writing—original draft preparation, X.T. and Y.H. All authors have read and agreed to the published version of the manuscript.

Funding: This study was funded by National Natural Science Foundation of China (61602214).

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare that there are no conflict of interest regarding the publication of this paper.

Abbreviations

The following abbreviations are used in this manuscript:

QoE	quality of experience
AMARL	attention multiagent reinforcement learning
CDN	content delivery network
EUA	edge user allocation
Dec-POMDP	decentralized partially observable Markov decision process
SFT	segment fetch time
RTT	round-trip time
DNN	deep neural network
MADDPG	multiagent deep deterministic policy gradient

References

1. Lai, P.; He, Q.; Abdelrazek, M.; Chen, F.; Hosking, J.; Grundy, J.; Yang, Y. Optimal edge user allocation in edge computing with variable sized vector bin packing. In Proceedings of the Service-Oriented Computing: 16th International Conference, ICSOC 2018, Hangzhou, China, 12–15 November 2018 ; Springer: Berlin/Heidelberg, Germany, 2018; pp. 230–245._15. [CrossRef]
2. Lai, P.; He, Q.; Cui, G.; Xia, X.; Abdelrazek, M.; Chen, F.; Hosking, J.; Grundy, J.; Yang, Y. Edge user allocation with dynamic quality of service. In Proceedings of the International Conference on Service-Oriented Computing, Toulouse, France, 28–31 October 2019; Springer: Berlin/Heidelberg, Germany, 2019; pp. 86–101. [CrossRef]
3. Wu, C.; Peng, Q.; Xia, Y.; Ma, Y.; Zheng, W.; Xie, H.; Pang, S.; Li, F.; Fu, X.; Li, X.; et al. Online user allocation in mobile edge computing environments: A decentralized reactive approach. *J. Syst. Archit.* **2021**, *113*, 101904. [CrossRef]
4. Mao, H.; Netravali, R.; Alizadeh, M. Neural adaptive video streaming with pensieve. In Proceedings of the Conference of the ACM Special Interest Group on Data Communication, Los Angeles, CA, USA, 21–25 August 2017; pp. 197–210. [CrossRef]
5. Panda, S.P.; Banerjee, A.; Bhattacharya, A. User Allocation in Mobile Edge Computing: A Deep Reinforcement Learning Approach. In Proceedings of the 2021 IEEE International Conference on Web Services (ICWS), Chicago, IL, USA, 5–11 September 2021; pp. 447–458. [CrossRef]
6. Nam, Y.; Chung, J.M. Cooperative content delivery for cost minimization in wireless networks. In Proceedings of the 2015 17th Asia-Pacific Network Operations and Management Symposium (APNOMS), Busan, Republic of Korea, 19–21 August 2015; pp. 566–568. [CrossRef]
7. Arumaithurai, M.; Seedorf, J.; Paragliela, G.; Pilarski, M.; Niccolini, S. Evaluation of ALTO-enhanced request routing for CDN interconnection. In Proceedings of the 2013 IEEE International Conference on Communications (ICC), Budapest, Hungary, 9–13 June 2013; pp. 3519–3524. [CrossRef]
8. Fang, C.; Yao, H.; Wang, Z.; Tu, Y.; Chen, Y. Edge Cache-based Intelligent Content Delivery in Information-Centric Wireless Networks. In Proceedings of the 2018 1st IEEE International Conference on Hot Information-Centric Networking (HotICN), Shenzhen, China, 15–17 August 2018; pp. 236–237. [CrossRef]
9. Lai, P.; He, Q.; Cui, G.; Chen, F.; Grundy, J.; Abdelrazek, M.; Hosking, J.; Yang, Y. Cost-effective user allocation in 5g noma-based mobile edge computing systems. *IEEE Trans. Mob. Comput.* **2021**, *21*, 4263–4278. [CrossRef]
10. Peng, Q.; Xia, Y.; Feng, Z.; Lee, J.; Wu, C.; Luo, X.; Zheng, W.; Pang, S.; Liu, H.; Qin, Y.; et al. Mobility-aware and migration-enabled online edge user allocation in mobile edge computing. In Proceedings of the 2019 IEEE International Conference on Web Services (ICWS), Milan, Italy, 8–13 July 2019; pp. 91–98. [CrossRef]
11. Cui, G.; He, Q.; Xia, X.; Chen, F.; Dong, F.; Jin, H.; Yang, Y. Ol-eua: Online user allocation for noma-based mobile edge computing. *IEEE Trans. Mob. Comput.* **2021**, *22*, 2295–2306. [CrossRef]
12. Birabwa, D.J.; Ramotsoela, D.; Ventura, N. Multi-agent deep reinforcement learning for user association and resource allocation in integrated terrestrial and non-terrestrial networks. *Comput. Netw.* **2023**, *231*, 109827. [CrossRef]
13. Li, Z.; Zhu, X.; Gahm, J.; Pan, R.; Hu, H.; Begen, A.C.; Oran, D. Probe and adapt: Rate adaptation for HTTP video streaming at scale. *IEEE J. Sel. Areas Commun.* **2014**, *32*, 719–733. [CrossRef]
14. Jiang, J.; Sekar, V.; Zhang, H. Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive. In Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies, Roma, Italy, 6–9 December 2012; pp. 97–108. [CrossRef]
15. Huang, T.Y.; Johari, R.; McKeown, N.; Trunnell, M.; Watson, M. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. In Proceedings of the 2014 ACM Conference on SIGCOMM, Chicago, IL, USA, 17–22 August 2014; pp. 187–198. [CrossRef]
16. Spiteri, K.; Urgaonkar, R.; Sitaraman, R.K. BOLA: Near-optimal bitrate adaptation for online videos. *IEEE/ACM Trans. Netw.* **2020**, *28*, 1698–1711. [CrossRef]
17. Liu, C.; Bouazizi, I.; Gabbouj, M. Rate adaptation for adaptive HTTP streaming. In Proceedings of the Second Annual ACM Conference on Multimedia Systems, Santa Clara, CA, USA, 23–25 February 2011; pp. 169–174. [CrossRef]
18. Liu, C.; Bouazizi, I.; Hannuksela, M.M.; Gabbouj, M. Rate adaptation for dynamic adaptive streaming over HTTP in content distribution network. *Signal Process. Image Commun.* **2012**, *27*, 288–311. [CrossRef]

19. Wu, X.; Li, X.; Tong, X.; Xie, R.; Song, L. Reinforcement Learning Based Adaptive Bitrate Algorithm for Transmitting Panoramic Videos. In Proceedings of the 2019 IEEE International Symposium on Circuits and Systems (ISCAS), Sapporo, Japan, 26–29 May 2019; pp. 1–5. [CrossRef]
20. Xiao, Q.; Ye, J.; Pang, C.; Ma, L.; Jiang, W. Adaptive Video Streaming via Deep Reinforcement Learning from User Trajectory Preferences. In Proceedings of the 2020 IEEE 39th International Performance Computing and Communications Conference (IPCCC), Austin, TX, USA, 6–8 November 2020; pp. 1–8. [CrossRef]
21. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.A.; Veness, J.; Bellemare, M.G.; Graves, A.; Riedmiller, M.; Fidjeland, A.K.; Ostrovski, G.; et al. Human-level control through deep reinforcement learning. *Nature* **2015**, *518*, 529–533. [CrossRef] [PubMed]
22. Nguyen, T.V.; Nguyen, N.P.; Kim, C.; Dao, N.N. Intelligent aerial video streaming: Achievements and challenges. *J. Netw. Comput. Appl.* **2023**, *211*, 103564. [CrossRef]
23. Lakew, D.S.; Tran, A.T.; Dao, N.N.; Cho, S. Intelligent Offloading and Resource Allocation in Heterogeneous Aerial Access IoT Networks. *IEEE Internet Things J.* **2023**, *10*, 5704–5718. [CrossRef]
24. Rashid, T.; Samvelyan, M.; Schroeder, C.; Farquhar, G.; Foerster, J.; Whiteson, S. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. *J. Mach. Learn. Res.* **2020**, *21*, 1532–4435. [CrossRef]
25. Hostallero, W.J.K.D.E.; Son, K.; Kim, D.; Qtran, Y.Y. Learning to factorize with transformation for cooperative multi-agent reinforcement learning. In Proceedings of the 31st International Conference on Machine Learning, Proceedings of Machine Learning Research, PMLR, Singapore, 16–18 April 2019.
26. Lowe, R.; Wu, Y.I.; Tamar, A.; Harb, J.; Pieter Abbeel, O.; Mordatch, I. Multi-agent actor-critic for mixed cooperative-competitive environments. *Adv. Neural Inf. Process. Syst.* **2017**, *30* , 1–12.
27. Nguyen, T.H.; Park, L. HAP-Assisted RSMA-Enabled Vehicular Edge Computing: A DRL-Based Optimization Framework. *Mathematics* **2023**, *11*, 2376 . [CrossRef]
28. Uhlenbeck, G.E.; Ornstein, L.S. On the theory of the Brownian motion. *Phys. Rev.* **1930**, *36*, 823. [CrossRef]