Edge-FVV: Free Viewpoint Video Streaming by Learning at the Edge

Haipeng Zhang[1,2], Jie Zhang[3], Weimiao Feng[4], Kaigui Bian[1,2], Hu Tuo[5]
[1] National Key Laboratory for Multimedia Information Processing, China, [2] Peking University, China
[3] University of Bath, UK
[4] Institute of Information Engineering of CAS, China
[5] IQIYI Science & Technology Co., Ltd., China
2201111598@pku.edu.cn, jz2558@bath.ac.uk, fengweimiao@iie.ac.cn, bkg@pku.edu.cn, tuohu@qiyi.com

***Abstract*—Audiences can gain an immersive experience watching videos from multiple angles (a.k.a. viewpoints). Free Viewpoint Video (FVV) is developed to enable users to choose their preferred viewpoints during the play of a video. However, users may experience a delay if video frames of the chosen viewpoint cannot be timely loaded, or synthesized from multiple video streams of neighboring viewpoints. To address this problem, we present Edge-FVV, an edge-assisted FVV system that employs edge caches to reduce the delay in streaming the requested FVV from the server to client users. We first analyze the capacity and delay at edge caches when answering FVV requests. Next, we propose two types of machine learning algorithms that allocate the users' requests to appropriate edge caches. Our evaluation shows that two types of proposed algorithms outperform benchmarks by 4.2-7.4% and 4.6-6.8%, respectively, in reducing the delay for FVV requests.**

***Index Terms*—Free viewpoint video, resource allocation, virtual view synthesis**

Fig. 1. An example of FVV viewed from four different viewpoints, where the dance is at the center of the stage and a number of cameras are capturing the videos of multiple viewpoints simultaneously.

I. Introduction

With increased bandwidth in communications networks, people can easily access high-definition video for immersive experiences, like Virtual Reality (VR) which creates a virtual environment with scenes that appear to be real. Another typical immersive video technology is Free Viewpoint Video (FVV), which records video content (e.g., dance, shows, and sports) through a set of cameras surrounding the scene. The users can watch the video from any angle (a.k.a. viewpoint) in a regular video player (e.g., web browser player), without any wearable devices (e.g., headsets).

An FVV system prescribes two important protocols: acquisition and virtual view synthesis. FVV acquisition is done by deploying a number of cameras to capture the scene from different viewpoints and then uploading captured videos to the server via multiple video streams (each stream representing the video from a viewpoint). If a user chooses to watch a viewpoint that has never been recorded by any camera, the system needs to perform a two-step virtual viewpoint synthesis operation. First, the user's device requests multiple video streams, neighboring the chosen viewpoint of the user as the reference viewpoints, from the server. Then, the user's device synthesizes the received video streams, produces a single video stream at the chosen viewpoint, and delivers it to the video player.

While FVV acquisition can be completed offline, the virtual view synthesis protocol may consume significant bandwidth and computing resources. To alleviate these burdens, existing research [1], [2] proposes adding extra edge caches (located between the server and the users) to undertake the virtual view synthesis task, given a requested viewpoint. The videos of reference viewpoints can be stored in the edge caches to accelerate the process for incoming requests to the given viewpoint. However, a challenge arises regarding the optimal number of caches. On one hand, more edge caches will distribute users' requests more evenly, leading to less delay for virtual view synthesis to serve each user. On the other hand, with more caches, each one is likely to have fewer reference viewpoints stored, increasing the delay of downloading the videos from the server if the requested reference viewpoint is unavailable in the cache.

In this paper, we present Edge-FVV, an edge-assisted FVV system that deploys a number of edge caches between the server and users to reduce the total delay for the FVV requests. First, we analyze the relationship between cache capacity and delay, and strike a balance between the delay for virtual view synthesis and that of downloading from the server, so as to minimize the total processing time for FVV requests. Next, based on different situations, we propose two types of algorithms that allocate the users' requests to appropriate edge caches. We evaluate their performance in reducing the total delay for FVV requests through extensive experiments. Our experimental results are promising, indicating that the proposed algorithm can effectively reduce the total delay of all the users over benchmarks by 4.2-7.4% and 4.6-6.8%, by choosing appropriate edge caches for the users.

II. Related Work

There have been many works in the field of transmission, content delivery, and other related areas for traditional and 360-degree videos [3]–[6]. However, the field of Free Viewpoint Video (FVV) remains largely unexplored. In this paper, we discuss the construction and optimization of a FVV system.

Building a distributed FVV system. Most existing works devote to improving virtual view synthesis methods and video encoding, and only a few works focus on building a real-world FVV system. A key problem in the designing of an FVV system is how to choose the location to conduct the virtual view synthesis. Most of the research assumes that the virtual view synthesis can be finished by the server or on the users' devices. But in practice, as the number of users increases, the growing number of virtual view synthesis tasks will place a huge burden on the server or cost extra computational resources on the users' devices. Hence, a distributed system is needed where the synthesis task is conducted at the network edge (e.g., proxy, edge cache, edge server) [1], [2]. A distributed FVV system can focus on improving the navigation quality for the client, and on studying the reference view selection problem under limited bandwidth [2]. The network architecture of an FVV system can be abstracted as a k-ary tree with the main server at the root and the users at the leaves, where the system calculates the total bandwidth cost to optimize the selected location of the edge servers [1]. In this work, we study the impact of the number of edge caches on the delay for a user's request, and seek to optimize the delay by allocating users to caches.

Virtual view synthesis methods. A popular method in virtual view synthesis is Depth Image Based Rendering (DIBR). Using pictures shot from different viewpoints along with corresponding depth images, DIBR methods can synthesize the picture from a virtual viewpoint by 3D image warping and inpainting [7]–[10]. Another popular method is to use Neural Networks to synthesize virtual viewpoints from pure images inputs [11]–[13] or reconstruct the 3D model of the scene [14]–[17]. However, the aforementioned methods are either time-consuming or limited in the form of input and output, which are not suitable for large-scale FVV systems. We turn our attention to Video Frame Interpolation (VFI) methods [18]–[21]. By using VFI methods the edge caches only need two adjacent reference viewpoints to synthesize a virtual view in Edge-FVV. Also, it is easy to extend Edge-FVV to support other synthesis methods that use more or less reference video streams.

III. Edge-FVV System

Edge-FVV architecture. Edge-FVV adopts a three-tier architecture that consists of the server, edge caches, and the users. Fig. 2 shows the architecture, in which the acquisition, and (virtual view) synthesis protocols are employed. Neither the server nor the users' devices are suitable for synthesis in large-scale FVV systems, due to the high memory and computation requirements of existing virtual view synthesis methods. Hence, Edge-FVV adds a tier of edge caches between the server and users. In this way, we pull the server and users from the synthesis tasks and save their costs.

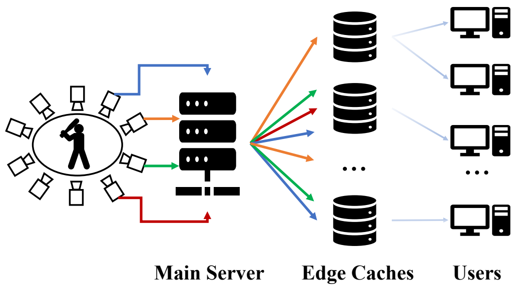


Fig. 2. The architecture of the Edge-FVV system.

VFI-based synthesis methods. In Edge-FVV, we choose a Video Frame Interpolation (VFI) method [21], which is easy to implement. Usually, VFI needs two consecutive frames in time as an input, and synthesizes the frames between them. However, in Edge-FVV, we consider the picture change caused by the change of viewpoints as the motion in videos, using VFI to supplement the missing viewpoints between two original reference pictures. With VFI, Edge-FVV only needs to transmit two reference video streams from the server to the edge caches, to synthesize a virtual viewpoint.

When to request viewpoints. In Edge-FVV, when a user chooses a specific viewpoint, it sends a request to a nearby edge cache. If the video stream of the requested viewpoint is already locally stored in the cache, then the stream is transmitted to the user; otherwise, the cache will request the user's viewpoint from the server.

When to conduct synthesis. If the server has not acquired the requested viewpoint, the edge cache finds two adjacent reference viewpoints of the requested viewpoint and downloads the reference video streams from the server. After the first pack of necessary frames is transmitted, the cache will start the synthesis task by rendering the requested viewpoint frames and then will transmit them to the user.

IV. Methodology

A. Relationship of Capacity and Delay

In Edge-FVV, the number of edge caches (denoted by $\mathcal{N}_c$) plays a dual role in the total delay of each user request. The more caches deployed, the fewer users are served by each cache, leading to less virtual view synthesis time for a cache to serve the users. However, if a cache serves fewer users, it is likely to have fewer number of reference viewpoints in the cache, which in turn, will increase the likelihood of downloading reference viewpoints from the server.

We denote by $\mathcal{V}$ the number of cameras in the acquisition protocol. That is, there are at most $\mathcal{V}$ original video streams generated by the cameras. For a specific virtual viewpoint, the edge cache needs to receive its left and right neighboring reference viewpoints to perform the synthesis. When an edge cache is synthesizing virtual views, according to users' requests, only part of those $\mathcal{V}$ original video streams are necessary as reference videos. These videos will be transmitted from the server to the edge cache. We denote the number of reference videos stored at the ith edge cache as m_i, and the expectation of m_i by $\mathcal{M}$. To calculate the processing time, we denote the total number of users by $\mathcal{U}$, and the number of users served by the ith edge cache by u_i. Suppose, for an original reference video stream, the edge cache needs to download a unit size of the file to start the synthesis task; we denote by $\mathcal{B}$ the edge cache's bandwidth. In the synthesis process, suppose an edge cache can concurrently perform $\mathcal{G}$ virtual view synthesis tasks, and the time an edge cache needs to finish one synthesis task is $\mathcal{T}_v$.

The processing time $\mathcal{T}$ for a user's request consists of two parts: the download time and the synthesis time. We denote them as $\mathcal{T} = t_d + t_s$, where t_d represents the download time and t_s represents the synthesis time. We present the following propositions.

The capacity of the edge cache. First, we need to figure out the capacity of the i-th cache, i.e., the number of reference videos stored at that cache. There are $\mathcal{V}$ gaps between the viewpoints of $\mathcal{V}$ original videos. For a single reference video stream v_k, it is used by a user if this user's virtual viewpoint falls into its left gap or right gap. Therefore the probability that v_k is used by a specific user is $\frac{2}{\mathcal{V}}$. In the i-th cache, for all the u_i users the cache is serving, the viewpoints they watch are independent of each other. Therefore, the probability that v_k is not used by any user in the i-th cache is $\mathcal{P}_{ni} = (1 - \frac{2}{\mathcal{V}})^{u_i}$ And the probability that v_k is in the i-th cache is $1 - \mathcal{P}_{ni}$. We can have the expectation of m_i as

$$\mathcal{M} = E\Big(\sum_{k=0}^{\mathcal{V}-1} v_k\Big) = \sum_{k=0}^{\mathcal{V}-1}\Big(E(v_k)\Big) = \mathcal{V}\bigg(1 - \Big(1 - \frac{2}{\mathcal{V}}\Big)^{u_i}\bigg) \tag{1}$$

The delay of video download and synthesis. Let t_{di} denote the delay for video caching at the i-th edge cache In the i-th cache, m_i video streams are already in transmission. When the cache wants to transmit n_i new video streams according to a new request, the new streams have to share the bandwidth equally with the other m_i video streams. We can have the download time $t_{di} = \frac{m_i + n_i}{\mathcal{B}}$.

For one virtual viewpoint synthesis task, two reference videos are needed. In the i-th cache, we denote the probability that one reference video is already cached and another reference video stream needs to be downloaded as $\mathcal{P}_{1i}$, and the probability that both the two reference video streams need to be downloaded as $\mathcal{P}_{2i}$. We can calculate the expectation of the downloading time as:

$$E(t_{di}) = \mathcal{P}_{1i} \cdot \frac{m_i + 1}{\mathcal{B}} + \mathcal{P}_{2i} \cdot \frac{m_i + 2}{\mathcal{B}} \tag{2}$$

For a new synthesis task, the probability that only one new stream needs to be downloaded is $\mathcal{P}_1 = 2(1 - \mathcal{P}_{ni})\mathcal{P}_{ni}$ The probability that two new streams need to be downloaded is $\mathcal{P}_2 = (\mathcal{P}_{ni})^2$

The expectation of the time needed to download new reference videos is

$$\begin{aligned} E(t_{di}) &= \mathcal{P}_{1i} \cdot \frac{m_i + 1}{\mathcal{B}} + \mathcal{P}_{2i} \cdot \frac{m_i + 2}{\mathcal{B}} \\ &= \frac{\mathcal{P}_{ni}(2(m_i + 1) - m_i \mathcal{P}_{ni})}{\mathcal{B}} \\ &= \frac{(\mathcal{V} - m_i)({m_i}^2 + 2\mathcal{V} + \mathcal{V}m_i)}{\mathcal{B}\mathcal{V}^2} \end{aligned} \tag{3}$$

When a new synthesis task comes to the i-th cache, we can estimate the time this task needs to wait is $\frac{u_i}{\mathcal{G}} \cdot \mathcal{T}_v$, so we can estimate the total synthesis time as

$$E(t_{si}) = \Big(\frac{u_i}{\mathcal{G}} + 1\Big) \cdot T_v \tag{4}$$

The total delay at the edge cache. Therefore the expectation of the processing time is

$$\begin{aligned} E(\mathcal{T}_i) &= E(t_{di}) + E(t_{si}) \\ &= \frac{\mathcal{P}_{ni}(2(m_i + 1) - m_i \mathcal{P}_{ni})}{\mathcal{B}} + \Big(\frac{u_i}{\mathcal{G}} + 1\Big) \cdot \mathcal{T}_v \\ &= \frac{(\mathcal{V} - m_i)({m_i}^2 + 2\mathcal{V} + \mathcal{V}m_i)}{\mathcal{B}\mathcal{V}^2} + \Big(\frac{u_i}{\mathcal{G}} + 1\Big) \cdot \mathcal{T}_v \end{aligned} \tag{5}$$

On average, the number of users each server serves is $E(u_i) = \frac{\mathcal{U}}{\mathcal{N}_c}$, and the expected number of m_i can be represented as $\mathcal{M} = \mathcal{V}\big(1 - (1 - \frac{2}{\mathcal{V}})^{\frac{\mathcal{U}}{\mathcal{N}_c}}\big)$. We replace the u_i and m_i in (5) with their expected numbers, and estimate the expectation of the processing time of a cache as

$$E(T) = \frac{(\mathcal{V} - \mathcal{M})(\mathcal{M}^2 + 2\mathcal{V} + \mathcal{V}\mathcal{M})}{\mathcal{B}\mathcal{V}^2} + \Big(\frac{\mathcal{U}}{\mathcal{G}\mathcal{N}_c} + 1\Big) \cdot \mathcal{T}_v \tag{6}$$

where $\mathcal{M} = \mathcal{V}(1 - (1 - \frac{2}{\mathcal{V}})^{\frac{\mathcal{U}}{\mathcal{N}_c}})$, and $\mathcal{B}$, $\mathcal{G}$, $\mathcal{V}$, $\mathcal{U}$, $\mathcal{T}_v$ depends on the scene.

Number of edge caches to be deployed. Eqt. (6) enables us to derive the expectation of the processing time for any number of edge caches. We note that, however, it is infeasible to derive a closed-form expression of the optimal number of edge caches due to the function composition embedded in the above equation. Nonetheless, the possible number of edge caches is limited and not large, so we can enumerate them and determine the number of edge caches to be deployed through experiments. As the corresponding results in Section V show, when the synthesis time $\mathcal{T}_v$ is small, the processing time will have a local minimum at a small number of edge caches. It is economical to deploy edge caches according to this local minimum, as the increment of edge caches may be expensive and entail longer delays in responding to users' requests. We present our discussion in Section V.

B. Machine Learning-based User Allocation

In Edge-FVV, the edge caches are located across different regions. We denote the edge caches that the i-th user can connect to as the agent set A_i of that user, and the size of

A_i is $\mathcal{N}_A$. There are two different ways of matching the i-th user's request with a nearby cache: (1) a distributed way allowing each user to autonomously connect to an agent in A_i; or (2) a centralized way assigning a user to an agent in A_i by the server who has the global information.

A distributed approach: Multi-armed bandit based allocation. Different edge caches incur different synthesis times and transmission times. Also, the spatial distribution of the users in physical regions leads to different loads on the edge caches. Hence, each user can choose an agent based on its location and the historic information of the agent. We implement four multi-armed bandit algorithms, ϵ-greedy, ϵ-decay, ϵ-first, and Upper Confidence Bound (UCB) [22], for user-side allocation, and the experimental results are shown in Section V.

A centralized approach: DQN-based allocation. When the server has the states of all the edge caches, upon the arrival of a new request at a given viewpoint, the server can either choose the cache with the least number of users or choose a cache that stores the demanded reference views. However, neither approach can balance the weight of the synthesis time and the download time. Moreover, the assignment may have a long-term influence on all other users. If an agent has many incoming users, the assignment of the current user to this agent will increase the waiting time for all the other incoming users and hence will have a greater delay for all the users in total. To estimate the long-term reward of an assignment, we calculate the Q values for each assignment using Deep Q-Network (DQN). The number of serving users and the number of reference videos cached in each cache are first fed into two separate neural networks (namely, UserNet and the VideoNet) to embed the user features and video features. The two features are then concatenated and fed into the ValueNet to generate the Q value for each cache. The Edge-FVV system is considered as a reinforcement learning environment. Each time a user's request comes, the server chooses an agent for this user. The server's choice is considered as its action to interact with the environment. The processing time of this request is considered as the reward given by the environment. In DQN, the Q values are updated as

$$Q(s_t, a_t) = r_t + \gamma Q'(s_{t+1}, \arg\max Q(s_{t+1}, a)), \tag{7}$$

where s_t is the state at time t, a_t is the action, and r_t is the reward. The second term in (7) calculates the value of the next state. This way, DQN can estimate the long-term reward of an action. In our implementation, the input state to the DQN is the number of users and the number of reference videos in each edge cache state. Different from (7), when a user's request arrives, the available agents are only a subset of all the actions. The value of the next state of the edge cache cannot be simply estimated as the Q value achieved by the best action, since the best action cannot be chosen in most cases. To better describe the value of the next state, we estimate the spatial distribution of users along with the agent sets, and calculate the value of the next state as the weighted sum of the maximum Q value of each agent set using the following equation:

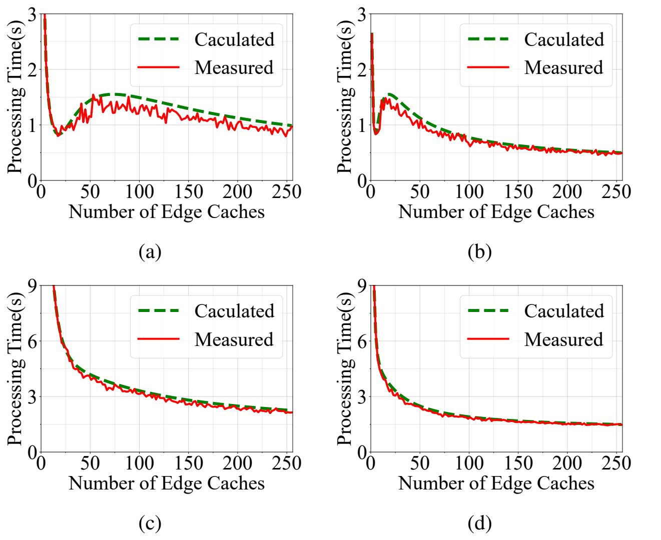


Fig. 3. Calculated and measured results of the average processing time using different numbers of edge caches, where $\mathcal{G} = 10$, $\mathcal{B} = 10$, $\mathcal{T}_v = 0.1$ and $\mathcal{U} = 1024$ for Fig. 3(a), $\mathcal{T}_v = 0.1$ and $\mathcal{U} = 256$ for Fig. 3(b), $\mathcal{T}_v = 1$ and $\mathcal{U} = 1024$ for Fig. 3(c),$\mathcal{T}_v = 1$ and $\mathcal{U} = 256$ for Fig. 3(d).

$$Q(s_t, a_t) = r_t + \gamma \sum_{A_i} P_{A_i} Q'(s_{t+1}, \arg\max_{a \in A_i}(Q(s_{t+1}, a))).$$

V. Evaluation

In this section, we evaluate Edge-FVV via simulations and experiments over a commercial FVV platform.

A. Impact of Number of Edge Caches

In Fig. 3, We plotted the curves of the delay of the FVV system with the change of the number of edge-caches under different conditions of the total number of users, bandwidth, GPU concurrency, and virtual viewpoint synthesis time based on the calculation and simulation results. The green dashed line and red solid line in the figure represent the calculation results and simulation results, respectively. In our simulation, we let the users individually choose their agents and viewpoints and then launch a new user request at a randomly chosen viewpoint.

We observe that the measured simulation results are slightly lower than the calculated processing time. This is because when a user requests a viewpoint, its left-hand side viewpoint and right-hand side viewpoint will be downloaded together and cached. Hence, the cached reference videos are not completely independent of each other. We also discover that the probability that the edge cache needs to apply for new viewpoint videos is slightly lower than we estimated under the independent assumption. However, our calculated result can effectively show the trend of the processing time change. From the results, we can tell that when the synthesis time is much smaller than the download time (e.g., as the results in Figs. 3(a) and 3(b)), the processing time curve will have a local minimum at an arbitrary number of edge caches. In such a case, it is economical to employ edge caches according to

TABLE I
COMPARISON OF DIFFERENT MULTI-ARMED BANDIT ALGORITHMS UNDER DIFFERENT SETTINGS. THE NUMBERS AFTER THE SLASH SHOWS THE RESULT ON THE COMMERCIAL PLATFORM.

Algorithms	$\mathcal{N}_c$=64 $\mathcal{N}_A$=16	$\mathcal{N}_c$=16 $\mathcal{N}_A$=4	10 ops	30 ops
ϵ-greedy	1.98	2.86/3.05	2.65/3.12	2.75/3.18
ϵ-first	**1.91**	**2.79/2.98**	**2.56/3.04**	**2.70**/3.08
ϵ-decay	1.94	2.80/2.98	2.58/3.07	2.72/**3.07**
UCB	2.08	2.90/3.17	2.66/3.22	2.81/3.29
random	2.11	2.97/3.22	2.67/3.26	2.89/3.38
fixed	2.11	2.97/3.17	2.66/3.23	2.83/3.29

the local minimum, and the increase in edge caches may lead to a higher processing time. When the download time is much smaller than the synthesis time (e.g., as shown in Figs. 3(c) and 3(d)), the processing time approximates the inverse ratio curve, and the effect of adding more edge caches diminishes as the number of servers increases.

B. Comparison of Multi-armed Bandit Algorithms

In the experiment, we emulate 128 users in total. We set $\mathcal{B}$ to 10, $\mathcal{G}$ to 10 and $\mathcal{T}_v$ to 1. In each time slot, we randomly choose a user to start watching FVV or change to a random viewpoint. The chosen user will select an edge cache to connect with, according to the employed multi-armed bandit algorithm. The connected edge cache will send back the measured processing time. The commercial platform provides us with the real distribution of users in different provinces in China mainland. We choose 16 locations of edge caches that are roughly evenly scattered in China, and select four agents for each user from the edge cache according to the distance of the users' geographical location.

Table I shows the processing time when comparing different multi-armed bandit algorithms on each user's device. We change the settings as shown in Table I, while in columns 1&2 we fix the average number of operations to 20 and in columns 3&4 we fix $\mathcal{N}_c$=16 and $\mathcal{N}_A$=4. Results show that all of the multi-armed bandit algorithms are better than letting the user randomly choose their agents or letting them connect to a fixed edge cache. In addition, ϵ-first and ϵ-decay outperform the others in the user allocation tasks. This implies that it is important to let users settle down to a fixed viewpoint when watching FVV, and the frequent change in users' agents will cause fluctuation in the processing time and confuse the multi-armed bandit algorithms. Note that in our implementation, ϵ-first only explores the agents in the first three operations, which is less than half the number of all the agents, but it still outperforms all the other algorithms. This shows that from the perspective of the processing time of all the users, to avoid fluctuation caused by exploration, the users do not have to find their best agent.

We also show results using different ϵ in ϵ-first and ϵ-greedy, and different rates of decay δ in ϵ-decay. In Table II, we set $\mathcal{N}_c = 32$ and $\mathcal{N}_A = 8$. The ϵ-greedy and ϵ-first are employed with $\epsilon = 0.1, 0.2, 0.5$, and ϵ-decay is employed with $\delta = 1, 0.5, 0.2$. The results show that the parameters have less effect on the average processing time of all the users, but the most explorative versions of each algorithm are still worse than the other ones. It is still harmful to let the users have too much exploration. We achieve the best performance using ϵ-first with $\epsilon = 0.2$.

TABLE II
COMPARISON OF USING DIFFERENT PARAMETERS IN MULTI-ARMED BANDIT ALGORITHMS.

Algorithms	Parameters	Average Processing Time
ϵ-greedy	ϵ=0.1, 0.2, 0.5	2.2170, 2.2137, 2.2246
ϵ-first	ϵ=0.1, **0.2**, 0.5	2.1755, **2.1664**, 2.1790
ϵ-decay	ϵ=1, 0.5, 0.2	2.2070, 2.2182, 2.2369

C. Evaluation of DQN

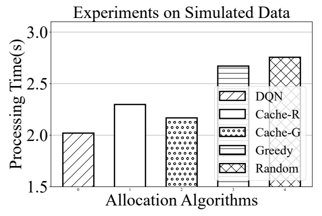
There are 16 edge caches in the system, and each user has 4 agents selected from the edge caches. In this way, we apply two sequential 16$\times$16 fully connected layers with ReLu as the activation layer both in the UserNet and the VideoNet to separately output two 16$\times$1 features for users and cached videos in each edge cache. The output of UserNet and VideoNet will be concatenated into a 16$\times$2 feature and input into QNet. We apply a 2$\times$16 fully connected layer with ReLU followed by a 16$\times$1 fully connected layer in the QNet to output Q values for each edge cache.

During the training process, we employ Double DQN, where one network will be used to estimate the Q values of the current state, and the algorithm will have another network that has the same architecture but different parameters to evaluate the Q values of the next state. The parameters of the second network will be copied from the first network every few iterations. In our implementation, we update the second network's parameters every 100 iterations. We set γ to 0.5 to consider the long-term influence of the main server's decisions. We use adam optimizer with a learning rate of 0.001. We also use epsilon-greedy with $\epsilon = 0.2$ to help our model to obtain experiences besides doing actions chosen by the model.

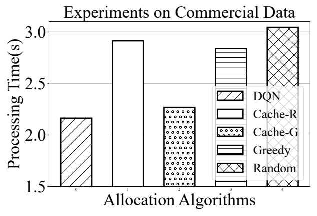
As shown in Fig. 4, we compare DQN to the algorithm that assigns the user to the edge cache with the fewest users (denoted as Greedy), the algorithm that randomly chooses an edge cache with the required reference videos in cache (denoted as Cache-R), and the algorithm that chooses the edge cache with the fewest users among all the edge caches that have the required videos in cache (denoted as Cache-G). The result shows that all the other algorithms suffer from balancing the weight of having the reference videos in the cache and the weight of having fewer users, while DQN can jointly consider the two factors as well as estimate the long-term reward of the assignment. Compared with Cache-G (which has the best performance among all the algorithms we compare with), the DQN reduces the total processing time by 4.6–6.8%.

VI. CONCLUSION

This paper presents Edge-FVV, an edge-assisted FVV System, where extra edge caches are added between the main



(a)



(b)

Fig. 4. Comparison of server-side allocation algorithms.

server and users to undertake the virtual view synthesis tasks. We first derive the mathematical relationship between the number of edge caches and the delay for an edge cache to serve a user's new viewpoint request, and figure out the number of edge caches to deploy that leads to a lower processing time. We then explore the allocation of users to edge caches in Edge-FVV by distributed and centralized algorithms. Experimental results show that the multi-armed bandit (distributed) and DQN-based (centralized) algorithms can effectively reduce the total processing time of all the users over benchmarks by 4.2-7.4% and 4.6-6.8% respectively, by choosing appropriate agents for the users.

Acknowledgement

This project is partially sponsored by National Key Research and Development Program of China No. 2020YFB2103801, NSFC 62032003, and Beijing Academy of Artificial Intelligence (BAAI). Jie Zhang was partially supported by a Leverhulme Trust Research Project Grant (2021 – 2024) and an EPSRC grant (EP/W014912/1).

References

[1] Á. Huszák, "Advanced free viewpoint video streaming techniques," *Multimedia Tools and Applications*, vol. 76, no. 1, pp. 373–396, 2017.

[2] L. Toni, G. Cheung, and P. Frossard, "In-network view synthesis for interactive multiview video systems," *IEEE Transactions on Multimedia*, vol. 18, no. 5, pp. 852–864, 2016.

[3] Y. Zhang, P. Zhao, K. Bian, Y. Liu, L. Song, and X. Li, "Drl360: 360-degree video streaming with deep reinforcement learning," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 1252–1260.

[4] Y. Zhang, K. Bian, H. Tuo, B. Cui, L. Song, and X. Li, "Geo-edge: Geographical resource allocation on edge caches for video-on-demand streaming," in *2018 4th International Conference on Big Data Computing and Communications (BIGCOM)*. IEEE, 2018, pp. 189–194.

[5] Y. Zhang, Y. Zhang, Y. Wu, Y. Tao, K. Bian, P. Zhou, L. Song, and H. Tuo, "Improving quality of experience by adaptive video streaming with super-resolution," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 1957–1966.

[6] Y. Guan, Y. Zhang, B. Wang, K. Bian, X. Xiong, and L. Song, "Perm: Neural adaptive video streaming with multi-path transmission," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 1103–1112.

[7] Z. Deng and M. Wang, "Reliability-based view synthesis for free viewpoint video," *Applied Sciences*, vol. 8, no. 5, p. 823, 2018.

[8] G. Luo and Y. Zhu, "Foreground removal approach for hole filling in 3d video and fvv synthesis," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 27, no. 10, pp. 2118–2131, 2016.

[9] A. Oliveira, G. Fickel, M. Walter, and C. Jung, "Selective hole-filling for depth-image based rendering," in *IEEE ICASSP*, 2015, pp. 1186–1190.

[10] A. Q. de Oliveira, T. L. da Silveira, M. Walter, and C. R. Jung, "A hierarchical superpixel-based approach for dibr view synthesis," *IEEE Transactions on Image Processing*, vol. 30, pp. 6408–6419, 2021.

[11] T. Habtegebrial, K. Varanasi, C. Bailer, and D. Stricker, "Fast view synthesis with deep stereo vision," *arXiv preprint arXiv:1804.09690*, 2018.

[12] J. Xie, R. Girshick, and A. Farhadi, "Deep3d: Fully automatic 2d-to-3d video conversion with deep convolutional neural networks," in *ECCV*. Springer, 2016, pp. 842–857.

[13] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "Nerf: Representing scenes as neural radiance fields for view synthesis," in *ECCV*. Springer, 2020, pp. 405–421.

[14] J. Chen, R. Watanabe, K. Nonaka, T. Konno, H. Sankoh, and S. Naito, "Fast free-viewpoint video synthesis algorithm for sports scenes," in *IEEE/RSJ IROS*. IEEE, 2019, pp. 3209–3215.

[15] J. Kilner, J. Starck, A. Hilton, and O. Grau, "Dual-mode deformable models for free-viewpoint video of sports events," in *Sixth International Conference on 3-D Digital Imaging and Modeling*. IEEE, 2007, pp. 177–184.

[16] G. Riegler and V. Koltun, "Free view synthesis," in *ECCV*. Springer, 2020, pp. 623–640.

[17] S. Rasmuson, E. Sintorn, and U. Assarsson, "A low-cost, practical acquisition and rendering pipeline for real-time free-viewpoint video communication," *The Visual Computer*, vol. 37, no. 3, pp. 553–565, 2021.

[18] S. Niklaus and F. Liu, "Softmax splatting for video frame interpolation," in *IEEE/CVF CVPR*, 2020, pp. 5437–5446.

[19] W. Bao, W.-S. Lai, C. Ma, X. Zhang, Z. Gao, and M.-H. Yang, "Depth-aware video frame interpolation," in *IEEE/CVF CVPR*, 2019, pp. 3703–3712.

[20] H. Jiang, D. Sun, V. Jampani, M.-H. Yang, E. Learned-Miller, and J. Kautz, "Super slomo: High quality estimation of multiple intermediate frames for video interpolation," in *IEEE/CVF CVPR*, 2018, pp. 9000–9008.

[21] Z. Huang, T. Zhang, W. Heng, B. Shi, and S. Zhou, "Rife: Real-time intermediate flow estimation for video frame interpolation," *arXiv preprint arXiv:2011.06294*, 2020.

[22] R. Agrawal, "Sample mean based index policies by o(log n) regret for the multi-armed bandit problem," *Advances in Applied Probability*, vol. 27, no. 4, p. 1054–1078, 1995.