Joint Communication and Computation Resource Allocation for Emerging mmWave Multi-User 3D Video Streaming Systems

Babak Badnava*, Jacob Chakareski†, Morteza Hashemi*
* Department of Electrical Engineering and Computer Science, University of Kansas
† College of Computing, New Jersey Institute of Technology

***Abstract*—We consider a multi-user joint rate adaptation and computation distribution problem in a millimeter wave (mmWave) virtual reality (VR) system. The VR system that we consider comprises an edge computing unit (ECU) that serves 360° videos to VR users. We formulate a multi-user quality of experience (QoE) maximization problem, in which VR users are assisted with the ECU to decode/render 360° videos. The ECU provides additional computational resources that can be used for processing video frames, at the expense of increased data volume and required bandwidth. To balance this trade-off, we leverage deep reinforcement learning (DRL) for joint rate adaptation and computational resource allocation optimization. Our proposed method, dubbed *Deep VR*, does not rely on any predefined assumption about the environment and relies on video playback statistics (*i.e.,* past throughput, decoding time, transmission time, etc.), video information, and the resulting performance to adjust the video bitrate and computation distribution. We train Deep VR with real-world mmWave network traces and 360° video datasets to obtain evaluation results in terms of the average QoE, peak signal-to-noise ratio (PSNR), rebuffering time, and quality variation. Our results indicate that the Deep VR improves the users' QoE compared to state-of-the-art rate adaptation algorithm. Specifically, we show a 3.08 dB to 4.49 dB improvement in video quality in terms of PSNR, a 12.5x to 14x reduction in rebuffering time, and a 3.07 dB to 3.96 dB improvement in quality variation.**

***Index Terms*—Quality of experience, mmWave network, 360° video streaming, edge computing, mobile VR systems.**

I. Introduction

It is envisioned that next generation wireless networks (6G-and-Beyond) will enable an unprecedented proliferation of computationally-intensive and bandwidth hungry applications, such as Augmented, Virtual, and Extended Reality (AR/VR/XR) [1**?** –3]. Emerging VR applications capture an entire spherical scene and stream high-fidelity 360° video content to create an immersive experience for VR users. This, in turn, requires high computational and communication resources. In particular, in contrast to traditional 2D video streaming, 360° video streaming requires computational resources for encoding, decoding, spatial processing, stitching, and rendering [4]. For instance, 360° video decoding entails spatio-temporal transformations for spherical projection. Viewport-adaptive streaming further increases the computational complexity by dynamically adjusting video segments based on the viewer's field of view (FoV). Moreover, 360° videos typically have higher resolution and larger file sizes compared to 2D videos, leading to higher bandwidth requirements. To satisfy these requirements, multi-access edge computing (MEC) and high-bandwidth mmWave networks have been used [4–7].

Although numerous studies have focused on maximization of QoE for 2D video streaming [8–10], increased computational

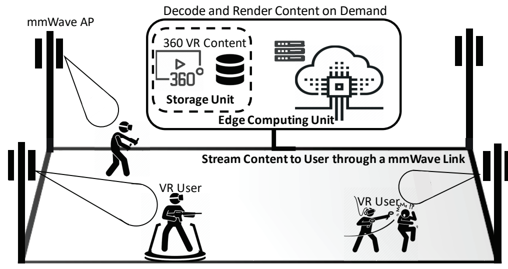

Decode and Render Content on Demand
mmWave AP
360 VR Content
360°
Storage Unit
Edge Computing Unit
Stream Content to User through a mmWave Link
VR User
VR User

Fig. 1: Edge-assisted VR system model: multiple VR headset connected to an edge computing unit through a mmWave network

and communication requirements call for joint resource allocation to provide a satisfactory QoE for VR users. Such a joint optimization problem should incorporate several constraints including: (i) *network-imposed* constraints that determine the available communication data rates, (ii) *computation-imposed* constraints that determine the available computational resources for VR users, (iii) *video-imposed* constraints that determine the *spatio-temporal characteristics* of 360° videos.

A multitude of prior studies have investigated various aspects of VR systems. For example, the authors in [4] propose a dual connectivity streaming system in which VR users leverage both mmWave and Wi-Fi alongside with an edge server to enable six degrees of freedom VR-based remote scene immersion. Hou et al. in [11] propose a FoV prediction algorithm, where the predicted view is encoded in relatively high quality and transmitted in advance to reduce latency. Hsu in [5] proposes a heuristic algorithm to optimize the caching and computing decision at an edge server to alleviate the network-imposed constraints and enhance users' QoE. Gupta et al. in [6] leverage an edge server for decoding to maximize the smallest immersion fidelity for the delivered 360° content across all VR users to alleviate the computation-imposed constraints. The authors in [7] present an MEC computing framework for AR applications to optimize energy efficiency and processing delay to alleviate the computation-imposed constraints. Moreover, [12] investigates the rate-distortion characteristics of ultra-high definition (UHD) 360° videos.

Despite extensive research on VR systems, to the best of our knowledge, no prior work has considered joint communication and computation resource allocation for mmWave-based multi-user VR systems. In particular, this paper formulates a multi-user edge-assisted QoE maximization framework for 360° video streaming on mmWave networks as depicted in Fig. 1. In this framework, the computational tasks (*i.e.,* decoding and rendering) necessary for processing 360° videos may be executed either by an ECU positioned near the VR arena or by the users' headsets themselves. On one hand, ECU has more computational resources, thus can process the 360° videos faster, which leads to a lower computation latency and a higher QoE for users. On the other hand, decoding and rendering at ECU introduces a higher bandwidth requirement for each user, which leads to higher communication latency since processed videos have much larger sizes, thus degrading the QoE. Furthermore, ECU provides its best performance for a certain number of users due to limited computational resources.

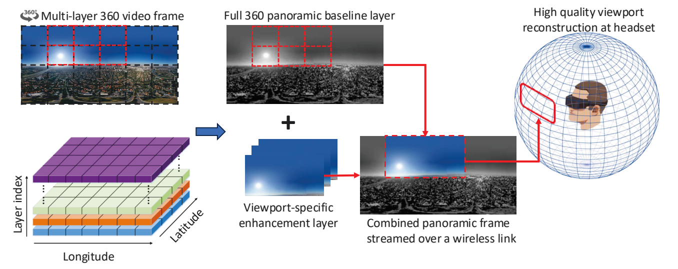
To jointly optimize communication and computation resource allocation, we present a novel learning-based decision-making algorithm, called Deep VR, that considers the interplay between the communication and computation requirements of 360° videos. Learning-based methods, particularly DRL, do not depend on predefined models or system assumptions. Rather, they learn to make resource allocation decisions exclusively by analyzing the outcomes of previous decisions and adapting accordingly. Deep VR leverages a state-of-the-art DRL method to learn the optimal computation distribution (*i.e.,* ECU or headset) and the video bitrate in the VR arena by considering the playback statistics and video information. In summary, the main contributions of this paper are as follows:

- We consider an edge-assisted VR streaming system model where an ECU provides VR users with 360° videos. We formulate a multi-user QoE maximization problem to find the best policy w.r.t network condition and spatio-temporal characteristics of multi-layer 360° videos.
- We develop a learning-based algorithm, called Deep VR, to find the optimal computation distribution and video bitrate for VR users to maximize their QoE. Our Deep VR agent observes the playback statistics (*i.e.,* past throughput, decoding/transmission time, etc.) and video information, then decides the optimal bitrate and computation distribution to decode/render the 360° video.
- We develop a gym-like [13] 360° VR streaming simulator using a real 360° video dataset, real-world VR user navigation information, and real-world mmWave network traces. Then, we perform an extensive simulation to analyze the behavior of our proposed method. We show that the proposed Deep VR algorithm improves the PSNR by 3.08 dB to 4.49 dB, rebuffering time by 12.5X to 14X, and the video quality variation by 3.07 dB to 3.96 dB.

II. Edge-Assisted VR System Model

We consider a multi-user 360° VR video streaming application with N users, which are connected to an ECU through a mmWave wireless access point. As depicted in Fig. 1, the users are equipped with VR headsets and request 360° videos that are stored on the ECU. The videos need to be prepared (*i.e.,* decoded/rendered) and transmitted to VR headset before users can play them. Upon receiving a request for a video segment, a decision-making agent (*i.e.,* Deep VR agent) makes a joint decision on the video bitrate allocated to each user and computation distribution (*i.e.,* decoding and rendering on the ECU or on the headset). Then, the computational resources will be allocated to prepare each video segment.

The ECU and all VR headsets are equipped with computational resources (*i.e.,* CPU and GPU) to prepare the video segments. If the decision-making agent decides to prepare the video segment on the ECU, the ECU's computational resources are shared among users to prepare the video segments and then transmit the video segments to the users. However, if the decision-making agent decides to prepare the video segment on users' headset, ECU sends the video segments to users and the preparation takes place at the headsets. Next, we present the models on multi-layer 360° videos, user headsets, and ECU.


Multi-layer 360 video frame
Full 360 panoramic baseline layer
High quality viewport reconstruction at headset
+
Layer index
Latitude
Longitude
Viewport-specific enhancement layer
Combined panoramic frame streamed over a wireless link
Fig. 2: Viewport-specific enhancement layers and a baseline layer are transmitted to a VR headset via a mmWave link to be recosntructed.

Multi-layer 360° Video Model: We consider the scalable multi-layer 360° video viewpoint tiling design [4]. As depicted in Fig. 2, each panoramic 360° video frame is partitioned into $\mathscr{L}$ tiles arranged in a $L_H \times L_V$ grid. A block of consecutive video frames, compressed together with no reference to others, creates a group of pictures (GoP) or video segment. Each video is divided into M GoP with fixed time duration of Δt, and L layers of increased immersion fidelity for each tile in a GoP exist. The first layer is called the base layer, and the remaining layers are denoted as enhancement layers.

Each enhancement layer increases the video bitrate, hence the video quality. We denote $e_m^n \in \{0,1\}^L$ as a one-hot vector that determines the number of enhancement layers included in m^{th} GoP requested by n^{th} user. Then, the size of the m^{th} compressed GoP tile, denoted by $d(e_m^n)$, is determined by summing over all tiles' bitrates in the user's viewport. We assume a positive compression reduction factor of $\beta < 1$, which leads to $d(e_m^n)/\beta$ for the size of the m^{th} decoded GoP tile. After decoding, GoPs need to be rendered as well, which leads to an increase in size by a factor of $\alpha \geq 2$. Hence, the size of the m^{th} GoP after decoding and rendering is determined by $\alpha d(e_m^n)/\beta$.

A. User Headset Model

The VR headsets are equipped with a CPU and GPU to process the videos, and each VR headset provides a maximum decoding speed of $\bar{Z}_{dec.}^n$, a maximum rendering speed of $\bar{Z}_{rend.}^n$, where n is the index of the VR headset. The VR headsets buffer the rendered videos in a buffer with fixed time duration length.

Buffer Dynamics: Fig. 3 illustrates the buffer dynamics of the 360° video streaming application. The GoPs need to be prepared in order to be buffered on headsets. At time t_m^n, the n^{th} user requests the m^{th} GoP. Then, the GoP will be prepared (*i.e.,* either on the ECU or the headset) and buffered on headsets. The preparation of m^{th} GoP involves three key stages: the decoding time D_m^n, rendering time P_m^n, and transmission time T_m^n. Once the GoP is prepared and buffered, the user waits for Δ_m^n seconds until requesting the next GoP. Thus, the next request time is: $t_{m+1}^n = t_m^n + D_m^n + P_m^n + T_m^n + \Delta t_m^n$. The buffer occupancy evolves as GoPs are being prepared, and the video is being played by the user. The buffer occupancy of user n increases by Δt

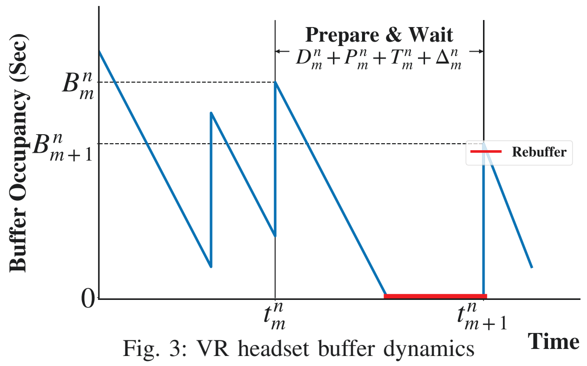


Fig. 3: VR headset buffer dynamics

seconds after receiving GoP m. Let $B_m^n = B^n(t_m^n)$ denote the buffer occupancy of the n^{th} user at t_m^n. Then, we have:

$$B_{m+1}^n = B^n(t_{m+1}^n) = \left((B_m^n - P_m^n - D_m^n - T_m^n)_+ + \Delta t - \Delta_m^n\right)_+ .$$

Here, the notation $(x)_+ = \max\{0,x\}$ ensures that the buffer occupancy is non-negative. If the preparation and transmission times take longer than the amount of GoP stored in the buffer (*i.e.,* $B_m^n < P_m^n + D_m^n + T_m^n$), then *rebuffering* happens as shown in Fig. 3. We also assume that the waiting time Δ_m^n is zero, except when the buffer is full, which the headset waits until the buffer has enough space to accommodate the next GoP, which leads to: $\Delta_m^n = \left((B_m^n - P_m^n - D_m^n - T_m^n)_+ + \Delta t - B_{max}^n\right)_+ .$

VR Headset Decoding and Rendering Model: We incorporate a decoding and rendering model developed by [2] to compute the decoding and rendering time for each GoP. In this model, the decoding time of the m^{th} GoP for the n^{th} user is assessed as $\tilde{si}(e_m^n)/\bar{Z}_{dec.}^n$, where $\tilde{si}(.)$ returns the decoding computational complexity of a GoP, in bits, which is the induced data rate associated with the current viewport and the quality of the streamed video, *i.e.,* $\tilde{si}(e_m^n) = d(e_m^n)$. Similarly, the rendering time is modeled as $si(e_m^n)/\bar{Z}_{rend.}^n$, where $si(.)$ returns the rendering computational complexity of a GoP, in bits, which is induced data rate after decoding of the GoP $si(e_m^n) = d(e_m^n)/\beta$. Note that we assume that the viewport information is available on the headset.

B. Edge Computing Unit (ECU) Model

The ECU provides an additional computational power to assist the VR users with decoding and rendering. This additional computational power provides a maximum decoding speed of $Z_{dec.}^{ECU}$ and a maximum rendering speed of $Z_{rend.}^{ECU}$, which is shared among the users that decode/render their GoP on the ECU. We assume that the decision-making time is negligible, since the decoding and rendering tasks are dominant overheads. Then, we incorporate a similar computation model on the ECU.

Video Decoding and Rendering Model: The decoding starts immediately after receiving the request for next GoP if the decision-maker decides to decode the GoP on the ECU. This leads to $\tilde{si}(e_m^n)/\Psi_m^n$ seconds of decoding time. Here, Ψ_m^n denotes the amount of decoding resources, out of $Z_{dec.}^{ECU}$, allocated to the n^{th} user for decoding the m^{th} GoP. Similarly, the ECU rendering time is modeled as $si(e_m^n)/\Theta_m^n$, where Θ_m^n denotes the amount of rendering resources, out of $Z_{rend.}^{ECU}$, allocated to the n^{th} user for the m^{th} GoP. Note that the total amount of computational resources allocated to users can not exceed the maximum available resources, which means $\sum_{n=1}^N \Psi_m^n \leq Z_{dec.}^{ECU}$ and $\sum_{n=1}^N \Theta_m^n \leq Z_{rend.}^{ECU}$ have to be satisfied for each GoP.

Communication Model: The ECU transmits GoPs through a mmWave wireless network. The expected transmission rate for a GoP is modeled as $C_m^n = \frac{1}{t_e - t_s}\int_{t_s}^{t_e} C_s^n ds$ where, t_s and t_e are transmission start and end times, respectively, and C_s^n is the throughput provided by the mmWave channel for the n^{th} user. Hence, the transmission time for a compressed GoP follows $d(e_m^n)/C_m^n$. Similarly, we can model the transmission time for decoded and rendered GoPs.

III. Multi-user QoE Maximization

Our goal is to improve QoE for multi-user 360° video streaming. To this end, we consider three major factors that impact the QoE. The first factor is the Average Video Quality (AVQ) defined as the average per-GoP video quality for tiles in user's FoV, expressed as follows: $\text{AVQ} = \frac{1}{M}\sum_{m=1}^M q(e_m^n)$. While, there are various choices for $q(.)$ [8], in this case, we use PSNR for the viewer's FoV [14], which can be calculated using the video distortion [12] as $q(e_m^n) = 10\log_{10}(255^2/MSE_m^n)$, where MSE_m^n is the distortion of the m^{th} GoP. The distortion has an inverse relation with the video bitrate, which is determined by the number of enhancement layers streamed to the user [12].

The second factor that impacts perceived QoE is the Average Quality Variation (AQV) that captures quality variation in user's FoV from one GoP to another. Therefore, we have $\text{AQV} = \frac{1}{M-1}\sum_{m=1}^{M-1}\left|q(e_{m+1}^n) - q(e_m^n)\right|$. The third factor that affects QoE is rebuffering. Rebuffering occurs if the preparation time of a GoP is larger than buffer occupancy level when the GoP was requested. Thus, the total rebuffering time (RT) is given by: $\text{RT} = \sum_{m=1}^M (D_m^n + P_m^n + T_m^n - B_m^n)_+$. A weighted sum of these factors defines user's QoE of a video with M GoPs as:

$$QoE_M^n = \text{AVQ} - \mu \times \text{RT} - \text{AQV}. \qquad (1)$$

Here, μ is a non-negative weighting parameter corresponding to the user sensitivity to rebuffering time. The QoE metric, defined in Eq. 1, allows us to model varying user preferences [8, 15].

QoE Maximization Problem: To formulate the problem of multi-user QoE maximization, we define two sets of communication and computation decision variables. In particular, $\phi_m^n \in \{0,1\}^3$ is a binary vector of size three with one active element (*i.e.,* a one-hot vector), which determines where the decoding and rendering take place for each GoP and user. There are three distinct states for ϕ_m^n:

$$\phi_m^n : \begin{cases} \phi_{m,0}^n = 1 & \Rightarrow \text{Decode \& Render on ECU,} \\ \phi_{m,1}^n = 1 & \Rightarrow \text{Decode on ECU \& Render on headset,} \\ \phi_{m,2}^n = 1 & \Rightarrow \text{Decode \& Render on headset.} \end{cases} \qquad (2)$$

For the sake of notation, we use $\phi_{m,i}^n$ instead of $\phi_m^n[i]$. In addition to computation location, we consider the rate allocation decision variable $e_m \in \{0,1\}^{N\times L}$ that determines how many enhancement layers should be streamed to each user in the VR arena. In addition to these decision variables, ECU computation resources should be allocated to the users who are determined to decode and/or render on the ECU. Therefore, ψ (*i.e.,* allocated decoding resources) and θ (*i.e.,* allocated rendering resources) need to be determined, forming the basis of the optimization problem outlined in Eq. 3.

$$
\begin{aligned}
\max_{\psi,\phi,\theta,\mathbf{e}} \quad & \mathbf{QoE} \\
\text{s.t.} \quad & B^n_{m+1} = \left((B^n_m - P^n_m - D^n_m - T^n_m)_+ + \Delta t - \Delta^n_m \right)_+ \\
& t^n_{m+1} = t^n_m \\
& \quad + \phi^n_{m,1} \left[\frac{\tilde{si}(e^n_m)}{\Psi^n_m} + \frac{si(e^n_m)}{\Theta^n_{m\cdot}} + \frac{\alpha d(e^n_m)/\beta}{C^n_m} \right] \\
& \quad + \phi^n_{m,2} \left[\frac{\tilde{si}(e^n_m)}{\Psi^n_m} + \frac{si(e^n_m)}{Z^n_{rend.}} + \frac{d(e^n_m)/\beta}{C^n_m} \right] \qquad (3) \\
& \quad + \phi^n_{m,3} \left[\frac{\tilde{si}(e^n_m)}{Z^n_{dec.}} + \frac{si(e^n_m)}{Z^n_{rend.}} + \frac{d(e^n_m)}{C^n_m} \right] + \Delta^n_m \\
& \sum_{n=1}^{N} \psi^n_m \le Z^{ECU}_{dec.} \quad \forall m \quad , \quad \sum_{n=1}^{N} \theta^n_m \le Z^{ECU}_{rend.} \quad \forall m
\end{aligned}
$$

There are several key challenges that need to be addressed before solving the optimization problem outlined in Eq. 3. This is a multi-objective optimization problem, in which we maximize the QoE for all the users, denoted by $\mathbf{QoE} = \left[QoE^1_M, QoE^2_M, ..., QoE^n_M\right]$. In general, multi-objective optimization problems have a set of so-called Pareto-optimal solutions, which implies that the QoE for one user can not be improved without degrading other users' QoE. Furthermore, the action space contains inter-dependent sub-actions. This implies that sub-actions put constraints on each other (*e.g.,* the maximum achievable rate is constrained by the rate adaptation decision). Another challenge arises from varying 360° video characteristics, which means the computational requirements may differ from one video to another. Finally, VR users experience time-varying and dynamic mmWave network condition (e.g., due to blockage, etc.) that impact the streaming data rate. Thus, a decision-making algorithm that incorporates various dynamic and time-varying conditions in terms of video quality, mmWave network, and available computational resources is desirable.

IV. Deep VR Framework

To tackle the challenges mentioned above, we present Deep VR framework. This framework comprises two main components: (i) A joint rate adaptation and computation distribution agent, which is modeled by a deep neural network, and (ii) a proportional resource allocation algorithm, which allocates decoding/rendering resources to users that have been distributed to ECU. Next, we describe the details of the proposed framework.

Deep VR Agent: At each time step and after receiving the request for each GoP from all users, the DRL agent observes the state s_m of all users. The state input s_m includes *playback statistics*, and *video information*. The video playback statistics include six critical metrics for each user to fully describe the 360° video playback status. These metrics are past throughput, past decoding time, past transmission time, past rendering time, last allocated rate e^n_{m-1}, and current buffer level B^n_m. We take the future GoPs size and number of remaining GoPs for each user as video information. This will help our Deep VR model to distinguish between videos with different spatio-temporal characteristics. All these metrics are collected for all users and stacked together to represent the state information.

Once the state s_m is observed, the agent chooses an action a_m to decode, render, and transmit GoPs. In this case, the Deep VR agent takes a joint action $a_m = (e_m, \phi_m)$. The rate allocation action $e_m \in \{0,1\}^{N \times L}$ determines how many enhancement layers

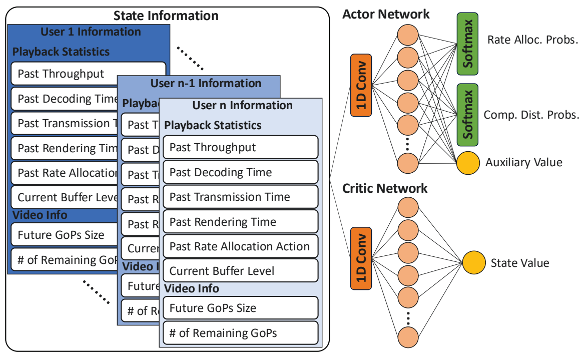


Fig. 4: Deep VR Architecture

will be streamed to each user. The computation distribution action $\phi_m \in \{0,1\}^{N \times 3}$ determines where each user decodes and/or renders the m^{th} GoP according to Eq. 2.

After taking the action, the state of the environment changes, and the agent receives a reward vector r_m that gives the reward for each user. The goal of our agent is to maximize the expected cumulative discounted reward for all users. We employ the change in users' perceived QoE at each step as the reward term. This is defined as $r^n_{m+1} = QoE^n_{m+1} - QoE^n_m$. This reward captures the changes in the perceived QoE as a result of the last action performed by the Deep VR agent, which enables the agent to learn the actions that lead to improvement in QoE.

Agent Architecture and Implementation: We develop a sample-efficient DRL algorithm to tackle the complexity of the defined multi-user QoE maximization problem. The Deep VR agent is composed of an actor network ω and a critic network ω_v. The actor network outputs the probabilities of both rate allocation action π^e_ω and computation distribution action π^ϕ_ω for all users. The actor network also outputs an auxiliary vector that estimates the state value for each user separately. The critic network outputs the estimated state value for each user separately. Inspired by [16, 17], we employ a two-phase training procedure, composed of a policy training phase and an auxiliary training phase [18]. In the policy training phase, the actor and critic networks are trained by Dual-Clip PPO [19]:

$$\mathscr{L}^{DClip} = \hat{\mathbb{E}} \left[\mathbb{1}(\hat{A}_m < 0) \max(\mathscr{L}^{PPO}, c\hat{A}_m) + \mathbb{1}(\hat{A}_m \ge 0) \mathscr{L}^{PPO} \right]. \quad (4)$$

Here, $\mathbb{1}()$ denotes a binary indicator function, and $\mathscr{L}^{PPO}$ represents the surrogate vanilla PPO loss that is:

$$\mathscr{L}^{PPO} = \mathscr{L}^{Clip}(\pi_\omega, \hat{A}_m) + \beta S_\omega(s_m) + \mathscr{L}^{Value}, \quad (5)$$

where $S(s_m)$ is the entropy of all policies, and β is the entropy weight, which jointly balance the tradeoff between exploration and exploitation during the learning process. $\mathscr{L}^{Value}$ is the value network loss [18], and $\mathscr{L}^{Clip}$ is the Single-Clip policy loss:

$$\mathscr{L}^{Clip} = \min \left[\rho(\pi_\omega, \pi_{\omega_{old}}) \odot \hat{A}_m, clip(\rho(\pi_\omega, \pi_{\omega_{old}}), 1 \pm \varepsilon) \odot \hat{A}_m \right]. \quad (6)$$

Here $\hat{A}_m = r_m + \gamma V_{\omega_v}(s_{m+1}) - V_{\omega_v}(s_m)$ is the advantage function that is calculated based on the current state-value estimate and the discount factor $\gamma = 0.99$, and $\odot$ is the element-wise multiplication. $\rho(\pi_\omega, \pi_{\omega_{old}}) = \left[\rho^1, \rho^2, \cdots, \rho^N\right]$ is a vector of size N that measures the changes in the new policy w.r.t. the old policy for all users (*i.e.,* the joint probability ratio of the

Algorithm 1 Deep VR Training Process

```
1: for epoch = $1,2,\ldots$ do
2:     Perform rollout under current policy $\pi$
3:     for $i = 1,2,\ldots,N_{Policy}$ do
4:         Optimize Eq. 4 (i.e., $\mathscr{L}^{DClip}$) w.r.t. $\omega, \omega_v$
5:     end for
6:     for $i = 1,2,\ldots,N_{aux}$ do
7:         Optimize $\mathscr{L}^{Joint}$ w.r.t. $\omega$
8:         Optimize $\mathscr{L}^{Value}$ w.r.t. $\omega_v$
9:     end for
10: end for
```

new policy to the old policy for the joint action):

$$\rho^i = \frac{\pi^{\phi}_{\omega}(\phi^i_m|s_m)}{\pi^{\phi}_{\omega_{old}}(\phi^i_m|s_m)} \frac{\pi^{e}_{\omega}(e^i_m|s_m)}{\pi^{e}_{\omega_{old}}(e^i_m|s_m)}.$$

In Eq. 6, $\rho(\pi_\omega, \pi_{\omega_{old}}) \odot \hat{A}_m$ measures the gained/lost reward as the policy for each user changes, which determines how a change in one user's policy affects the reward of other users. In the auxiliary phase, we further optimize the actor and critic networks according to a joint objective function, $\mathscr{L}^{Joint}$, which is composed of a behavioral cloning loss and an auxiliary value loss. Due to space limitation, details on auxiliary training phase are omitted. Interested readers can refer to [18].

Algorithm 1 presents the training process of Deep VR agent, which continues for multiple iterations until convergence. Each iteration is composed of three phases. In the first phase, we perform the current policy π_ω on a randomized environment to collect new experiences (*i.e.,* rollout process). We encapsulated our edge-assisted VR system into a gym-like environment, which allows the Deep VR agent to interact with the system effectively. The Deep VR agent learns the policy through its interaction with the environment. Both users' video and network condition are randomized in this environment so that the agent learns the optimal computation distribution and rate adaptation policy for various videos and network conditions. In the second phase, we update both actor and critic networks. We compute the Dual-Clip PPO loss $\mathscr{L}^{DClip}$, and use newly collected experiences (*i.e.,* resulted from the rollout process) to update the networks. Finally, in the third phase, we use all collected experiences to update both actor and critic networks by optimizing the behavioral cloning and value losses.

V. Evaluation

Baselines: In this section, we evaluate our proposed framework through an extensive simulation against Pensieve [8], a state-of-the-art rate adaptation algorithm. Pensieve is designed for 2D video streaming applications, and only adjust the video rate based on user state, while our problem is a joint computation distribution and rate adaptation algorithm. Thus, we employ two variants of Pensieve, namely ECU-Pensieve and Headset-Pensieve. ECU-Pensieve performs all the computations (*i.e.,* decoding and rendering) on the ECU, while Headset-Pensieve performs all the computations on the users' headset. Both use the original Pensieve with no modification to adaptively adjust the users' bitrates. Moreover, we modify our Deep VR framework and present two rate adaptation algorithms, ECU-R and Headset-R. ECU-R and Headset-R use a neural network with the same architecture as shown in Fig. 4 for rate adaptation, except that the computation distribution is not decided by the neural network and all the computations are performed on the ECU or headsets, respectively.

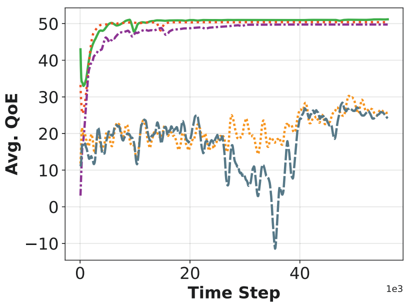
Metric / Baseline	PSNR [dB]	RT [sec]	QV [dB]
Deep VR	**52.91 ± 0.89**	**0.06 ± 0.04**	**1.26 ± 0.40**
ECU-R	52.66 ± 1.21	**0.06 ± 0.04**	1.28 ± 0.38
Headset-R	51.86 ± 1.71	**0.06 ± 0.04**	1.34 ± 0.44
ECU-Pensieve	49.83 ± 2.38	0.86 ± 1.50	4.33 ± 1.78
Headset-Pensieve	48.42 ± 1.69	0.75 ± 1.41	5.22 ± 1.09

TABLE I: Deep VR Performance during Testing Stage

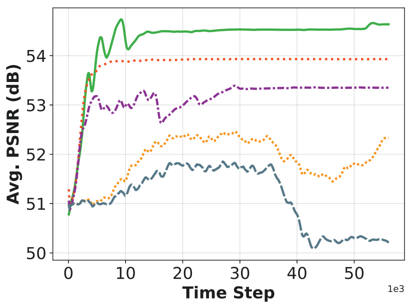
Datasets: In our simulations, we employ a full UHD 360° video dataset [12]. This dataset includes 15 videos with various spatio-temporal characteristics. Each video is represented using the multi-layer 360° model presented in Section II, and video frames are partitioned into an 8×8 grid. Bitrate information for seven layers, each offering progressively higher levels of immersion fidelity for each tile, is provided. Additionally, head movement data for multiple users is included, allowing us to determine the viewport location for each user. Moreover, we use a dataset of mmWave network throughput traces [20], which were collected in two different cities in the U.S. and from commercial operators (T-Mobile and Verizon).

Training Performance: We perform our experiments by setting $N = 6$ VR users, ECU decoding speed to $Z^{ECU}_{dec.} = 7.5$ Gbps, ECU rendering speed to $Z^{ECU}_{rend.} = 20$ Gbps, headsets' decoding and rendering speeds to $Z^{n}_{dec.} = 0.2$ Gbps and $Z^{n}_{rend.} = 9.4$ Gbps, respectively. We set $\mu = 30$ in Eq. 1, which is inspired from [8] and fine-tuned to provide a rebuffering time of less than two seconds. Fig. 5 shows the moving average performance of the Deep VR agent and the baselines over $2,000$ training episodes. The Deep VR agent learns to maximize QoE, which leads to maximizing the PSNR and minimizing the rebuffering time and quality variation. The Deep VR agent outperforms the Pensieve, which is due to its ability to distribute the computation between headsets and ECU in an environment with time-varying network condition. In fact, the Deep VR agent learns to distribute the computations on headsets for those users who have lower computational requirements. This, in turn, frees up resources on the ECU to allocate to those users with higher computational requirements, which exceed their headsets' resources.

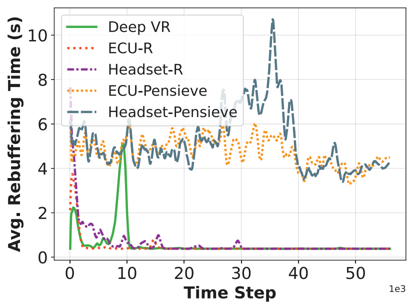
Deployment Performance: Fig. 6 demonstrates the trade-off between rebuffering time and PSNR. Each point demonstrates the average rebuffering time and PSNR experienced by users, and the vertical and horizontal bars represent the standard deviation of the rebuffering time and PSNR, respectively. A small rebuffering time and high PSNR with small variation is desirable, which is represented by a point in the lower right corner of this graph. Fig. 6 shows that the Deep VR agent achieves the smallest rebuffering time and the highest PSNR. Furthermore, Table I reports the average and standard deviations of PSNR, rebuffering time, and quality variation. The Deep VR agent demonstrates an improvement of 3.08 dB to 4.49 dB in PSNR, a 12.5x to 14x improvement in rebuffering time, and a 3.07 dB to 3.96 dB improvement in quality variation.



(a) Average user QoE



(b) Average PSNR perceived by users



(c) Average rebuffering time

Fig. 5: The Deep VR agent learns to maximize the QoE (*i.e.,* weighted average of PSNR, rebuffering time, and quality variation) in a VR arena with $N = 6$ users over 2000 episodes. The Deep VR agent outperforms two Pensieve variations due to its computation distribution ability.

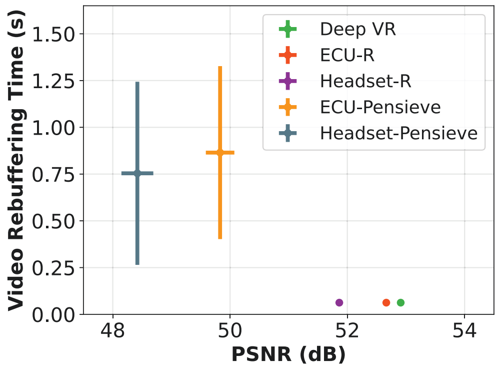


Fig. 6: Training performance of Deep VR compared to proportional resource allocation at ECU and headset Computation.

VI. Conclusion

In this paper, we considered the problem of multi-user joint rate adaptation and computation distribution in a VR arena for a 360° video streaming platform, where a learning-based agent decides on the video bitrate allocated to each user and computation distribution (*i.e.,* whether each video segment should be decoded/rendered on the ECU or on the headset). The overall objective is to maximize multi-user QoE under dynamic and time-varying conditions in terms of video requests, available computational resources, and communication bandwidth. Leveraging the state-of-the-art DRL algorithm, we developed Deep VR that utilizes playback statistics and video information to make a joint rate adaptation and computation distribution decision. Through numerical simulation using real-world network traces and 360° video information, we showed that the Deep VR agent learns to balance the existing trade-offs in the system and outperforms the state-of-the-art rate adaptation algorithm. Specifically, the Deep VR agent demonstrates a 3.08 dB to 4.49 dB improvement in PSNR, a 12.5x to 14x reduction in rebuffering time, and a 3.07 dB to 3.96 dB improvement in quality variation. Our current solution relies on the number of users in the system. In the future, we will investigate the effect of increasing number of users on the provided solution.

VII. Acknowledgement

This work was supported in part by NSF grants 1955561, 2212565, 2323189, 2032033, 2106150, and 2346528. The work of Jacob Chakareski was additionally supported in part by NIH award R01EY030470; and by the Panasonic Chair of Sustainability at NJIT.

References

[1] SeaGate, “State of the Edge,” https://www.seagate.com/www-content/enterprise-storage/it-4-0/images/Data-At-The-Edge-UP1.pdf, 2019, [Online].

[2] J. Chakareski, M. Khan, T. Ropitault, and S. Blandino, “Millimeter Wave and Free-Space-Optics for Future Dual-Connectivity 6DOF Mobile Multi-User VR Streaming,” *ACM Trans. Multimedia Comp. Comm. App.*, 2023.

[3] Statista, “AR and VR: Market data & analysis,” https://www.statista.com/outlook/amo/ar-vr/worldwide, [Online].

[4] J. Chakareski, M. Khan, T. Ropitault, and S. Blandino, “6DOF Virtual Reality Dataset and Performance Evaluation of Millimeter Wave vs. Free-Space-Optical Indoor Communications Systems for Lifelike Mobile VR Streaming,” in *Proceedings of 54th ACSSC*, 2020.

[5] C.-H. Hsu, “MEC-Assisted FoV-Aware and QoE-Driven Adaptive 360° Video Streaming for Virtual Reality,” in *Proceedings of 16th Intern. Conf. on Mobility, Sensing and Networking (MSN)*, 2020.

[6] S. Gupta, J. Chakareski, and P. Popovski, “mmWave Networking and Edge Computing for Scalable 360° Video Multi-User Virtual Reality,” *IEEE Transactions on Image Processing*, 2023.

[7] J. Ren, Y. He, G. Huang, G. Yu, Y. Cai, and Z. Zhang, “An Edge-Computing Based Arch. for Mobile Aug. Reality,” *IEEE Network*, 2019.

[8] H. Mao, R. Netravali, and M. Alizadeh, “Neural Adaptive Video Streaming with Pensieve,” in *Proc. ACM SIGCOMM*, 2017.

[9] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman, “BOLA: Near-Optimal Bitrate Adaptation for Online Videos,” *IEEE Trans. on Networking*, 2020.

[10] B. Badnava, S. Reddy Chintareddy, and M. Hashemi, “QoE-Centric Multi-User mmWave Scheduling: A Beam Alignment and Buffer Predictive Approach,” in *Proc. IEEE ISIT*, 2022.

[11] X. Hou, S. Dey, J. Zhang, and M. Budagavi, “Predictive Adaptive Streaming to Enable Mobile 360-Degree and VR Experiences,” *IEEE Trans. on Multimedia*, 2021.

[12] J. Chakareski, R. Aksu, V. Swaminathan, and M. Zink, “Full UHD 360-Degree Video Dataset and Modeling of Rate-Distortion Characteristics and Head Movement Navigation,” in *Proc. ACM Multimedia Sys.*, 2021.

[13] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, “Openai gym,” *arXiv:1606.01540*, 2016.

[14] M. Yu, H. Lakshman, and B. Girod, “A Framework to Evaluate Omnidirectional Video Coding Schemes,” in *Proceedings of 2015 IEEE International Symposium on Mixed and Augmented Reality*, 2015.

[15] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli, “A Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP,” in *Proceedings of the 2015 ACM Conf. on Special Interest Group on Data Comm.*, 2015.

[16] D. Ye, Z. Liu, M. Sun, B. Shi, P. Zhao, H. Wu, H. Yu, S. Yang, X. Wu, Q. Guo, Q. Chen, Y. Yin, H. Zhang, T. Shi, L. Wang, Q. Fu, W. Yang, and L. Huang, “Mastering Complex Control in MOBA Games with Deep Reinforcement Learning,” in *Proceedings of The 34 AAAI Conf.*, 2020.

[17] C. W. L. S. Tianchi Huang ; Chao Zhou ; Rui-Xiao Zhang, “Buffer Awareness Neural Adaptive Video Streaming for Avoiding Extra Buffer Consumption,” in *Proceedings of IEEE INFOCOM Conf.*, 2023.

[18] K. Cobbe, J. Hilton, O. Klimov, and J. Schulman, “Phasic Policy Gradient,” *arXiv:2009.04416*, 2020.

[19] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal Policy Optimization Algorithms,” *arXiv:1707.06347*, 2017.

[20] A. Narayanan, X. Zhang, R. Zhu, A. Hassan, S. Jin, X. Zhu, X. Zhang, D. Rybkin, Z. Yang, Z. M. Mao, F. Qian, and Z.-L. Zhang, “A Variegated Look at 5G in the Wild: Performance, Power, and QoE Implications,” in *Proceedings of ACM SIGCOMM*, 2021.