360-degree Video Streaming by Deep Reinforcement Learning

Aodong Li
New York University
Manhattan, NY
aodong.li@cs.nyu.edu

Liyang Sun
New York University
Brooklyn, NY
ls3817@nyu.edu

Chenge Li
New York University
Brooklyn, NY
cl2840@nyu.edu

Yao Wang
New York University
Brooklyn, NY
yw523@nyu.edu

Yong Liu
New York University
Brooklyn, NY
yongliu@nyu.edu

ABSTRACT

Current systems on 360-degree video streaming are usually specific to local video contents and local network conditions. Thus, failing to consider the effects on the far future decisions, these streaming systems are by no means optimal. Recently, the development of deep reinforcement learning (DRL) motivates its application in various domains. In video streaming, the sequential behavior of choosing appropriate bitrates makes it fit well into the reinforcement learning framework.

In this paper, we use DRL to perform 360-degree video streaming. We first notice some special cases have remarkable optimal decision patterns. Using this knowledge, we regularize the neural network with residual connections to improve the performance. Then we use synthetic special cases to illustrate our system's potential to make approximately optimal decisions, for which both layered coding and non-layered coding scenarios are investigated. Moreover, when we provide the knowledge of the Field of Vision (FoV) prediction accuracy for the streaming agent, higher performance is reached. Finally, we compare our proposed streaming agent with an advanced video streaming baseline in a practical 5G environment setting. To evaluate performances, We let both systems perform a 45-second streaming task with a uniform Quality of Experience (QoE) metric. Experiments show our proposed agent is consistently better: the performance is improved by 18.8% on average over the baseline.

1 INTRODUCTION

The emergence of Virtual Reality (VR) posits challenges for streaming techniques. They involve 1) video chunks are larger compared with a traditional video of the same length; 2) the bandwidth are limited; 3) VR demands for streaming in real time to gain a better Quality of Experience (QoE). Tackling these challenges, 360-degree multi-buffer streaming techniques[1][11] are usually used.

The main idea of these streaming methods is as follows. First use a relatively small portion of bandwidth to fill in a base layer/buffer, which ensures real-time playback but with basic video quality. Then coupled with other user's Field of Vision (FoV) prediction techniques, download a specific tile of the video to fill in an enhancement layer/buffer that will probably covers the user vision in the future. Obviously, many hyper-factors in the process may affect the streaming performance. These involves but not limited to: the choice of video coding method, the FoV prediction accuracy, most importantly, the potential of the video streaming agent. In this work, we mainly focus on video streaming agent but only introduce the other techniques when necessary.

One feature of the video streaming agent for 360-degree video streaming is that it is required not only to capture the dynamics of the bandwidth[1] but also to manage the interplay between different video buffers. Besides, sometimes it also needs to incorporate the information of the FoV accuracy to make a better decision. For example, if the accuracy is low, it would be better to fetch a base layer chunk to reduce the waste of bandwidth; on the other hand, if the accuracy is high, then an enhancement layer chunk would be good to the QoE. To handle such a complex task, a delicate streaming agent has to be designed. But it is hard for humans to search such a solution and encode rules. Intuitively, an agent that is able to learn the complicated relations would be desirable.

In this paper, motivated by the successful deployment of deep reinforcement learning (DRL) on traditional video streaming[8], we propose a system to perform more challenging 360-degree video streaming. First we investigate what an agent's decision behavior should be to render a higher QoE. Remarkably, we find an iterative fetching pattern between base layer and enhancement layer, implying that the last decision will highly influences the current decision to make. If we put last decision as an input state, then there is a direct relation between input and output. However, it is known that a perceptron with a nonlinear activation function is hard to model an identity mapping[5]. With this insight, we regularize our agent's network with residual connections from input to the other end. Experiments demonstrate our network can achieve optimal decisions for some artificially designed cases. Finally, we introduce the information of FoV prediction accuracy into the learning agents to further improve the decision quality. And we empirically show that our proposed system outperforms a baseline system[11], which uses PI controller to control the interplay between video layers.

The remaining of the paper is structured as follows: the next section briefly describes the background of 360-degree video streaming and reinforcement learning, and section 3 reviews some related work. Section 4 introduces the DRL 360-degree video streaming agent, and section 5 presents the experiments. Finally, the paper ends up with future work and conclusion.

[1]For a consistent discussion, we use the term bandwidth to indicate network bandwidth and term network for neural network, which is the streaming agent model structure.

2 BACKGROUND

2.1 360-degree Video Streaming

As we described in section 1, 360-degree video streaming is accomplished by multi-layer video coding technique. Layers have different functions, and we usually categorize them into two general classes: base layer and enhancement layer. Aiming for robustness, base layer's function is to ensure videos can be played smoothly without rebuffering and black spot. Hence for this layer, videos are coded with lower bitrates to cover the whole 360-degree vision and to endure bad bandwidth. On the other hand, aiming for high resolution, video chunks in enhancement layer are encoded with higher bitrates but may only cover a small portion of the video.

To be transmitted from streaming server to client, video is coded into chunks with different bitrates. Higher bitrate coding implies higher video resolution, i.e., higher video quality. Concretely, the relationship between bitrate and video chunk sizes can be understood the following way. Suppose the bitrate is R Mbps and the chunk length is t seconds, then the chunk size is roughly $Rt/8$M bytes. During video playback, it is the coded chunks that are sent across the network media. Streaming agent has to make decisions to select chunks with proper bitrates based on current environment.

Based on the relationship between base layer and enhancement layer, coding schemes are classified into layered coding and non-layered coding. Layered coding requires enhancement layer is a complement of base layer and enhancement layer can not be played without base layer. Non-layered coding, on the other hand, does not assume this relationship and it can be played without base layer. Usage of redundant coding bits in non-layered coding makes enhancement layer independent of base layer, but black vision spot possibly happens if base chunks are absent. Intuitively, trade-offs exist between these two coding schemes and different scenarios favor different coding scheme.

Streaming agents can generally work with different techniques. In this paper we tackle this task through DRL.

2.2 Reinforcement Learning

In this section, we briefly introduce the learning framework we use for this task.

Reinforcement learning has several methods to maximize the cumulative rewards. This paper focuses on one of them – policy optimization. Policy is the strategy that agents utilize to take actions. In mathematics, a stochastic policy $\pi(a|s)$ is a function that takes previous states as input and generates a probability distribution of actions.

Optimizing policy $\pi(a|s)$ is equivalent to finding one that maximizes the expected cumulative rewards of trajectories,

$$\mathrm{E}[\sum_{t=0}^{T-1} r_t|\pi]. \tag{1}$$

A trajectory refers to a sequence of states and actions $(s_0, a_0, s_1, a_1, ..., s_{T-1}, a_{T-1})$.

When the policy is parameterized by θ, we are equivalently solving

$$\max_\theta \mathrm{E}[\sum_{t=0}^{T-1} r_t|\pi_\theta] = \max_\theta \sum_{t=0}^{T-1} \mathrm{E}[r_t|\pi_\theta]$$

To learn parameter θ, gradient-based updating algorithms are usually used [12]. The main step is to compute the gradient of the sum of the expectations,

$$\nabla_\theta \sum_{t=0}^{T-1} \mathrm{E}[r_t|\pi_\theta] = \sum_{t=0}^{T-1} \nabla_\theta \mathrm{E}[r_t|\pi_\theta]. \tag{2}$$

The linearity of expectation tells us finding the gradient of each reward is enough. Then each step-wise expected reward $r_t, 0 \le t \le T-1$, which is a function of $\tau = (s_0, a_0, s_1, a_1, ..., s_t, a_t)$ by Markov decision process, can be represented in integral form

$$\mathrm{E}[r_t|\pi_\theta] = \int_\tau r_t(\tau) P_\theta(\tau) d\tau, \tag{3}$$

where $P_\theta(\tau)$ can be further expanded by chain rule into

$$P_\theta(\tau) = \mu_0(s_0)\pi_\theta(a_0|s_0)P(s_1|s_0, a_0)\pi_\theta(a_1|s_1)...P(s_t|s_{t-1}, a_{t-1})\pi_\theta(a_t|s_t)$$

with $\mu_0(s_0)$ being the initial distribution of s_0.

Taking the gradient to (3) we have

$$\begin{aligned}
\nabla_\theta \mathrm{E}[r_t|\pi_\theta] &= \nabla_\theta \int_\tau r_t(\tau) P_\theta(\tau) d\tau \\
&= \int_\tau r_t(\tau) \nabla_\theta P_\theta(\tau) d\tau \\
&= \int_\tau r_t(\tau) (\nabla_\theta \log P_\theta(\tau)) P_\theta(\tau) d\tau \\
&= \int_\tau r_t(\tau) \left(\nabla_\theta \sum_{t'=0}^{t} \log \pi_\theta(a_{t'}|s_{t'}) \right) P_\theta(\tau) d\tau \\
&= \mathop{\mathrm{E}}_{\tau \sim P_\theta(\tau)} \left[r_t(\tau) \left(\sum_{t'=0}^{t} \nabla_\theta \log \pi_\theta(a_{t'}|s_{t'}) \right) \right]
\end{aligned}$$

where the derivative utilizes the equality $\nabla_\theta P_\theta(\tau) = (\nabla_\theta \log P_\theta(\tau)) P_\theta(\tau)$ and the fact that when differentiating with respect to θ, unrelated terms drop out.

So we can rewrite the objective function 2 as

$$\sum_{t=0}^{T-1} \nabla_\theta \mathrm{E}[r_t|\pi_\theta] = \mathop{\mathrm{E}}_{\tau \sim P_\theta(\tau)} \left[\sum_{t=0}^{T-1} \left(r_t(\tau) \left(\sum_{t'=0}^{t} \nabla_\theta \log \pi_\theta(a_{t'}|s_{t'}) \right) \right) \right] \tag{4}$$

$$= \mathop{\mathrm{E}}_{\tau \sim P_\theta(\tau)} \left[\sum_{t=0}^{T-1} \left(\nabla_\theta \log \pi_\theta(a_t|s_t) \left(\sum_{t'=t}^{T-1} r_{t'}(\tau) \right) \right) \right]. \tag{5}$$

The last formula (5) provides a convenient way to approximate the policy gradient: sample a trajectory, compute the empirical gradient $\hat{g}$ by 5, then update θ through

$$\theta := \theta + \delta \hat{g}$$

where δ is the learning rate. Furthermore, one can update the parameter with more sophisticated optimizing algorithms like Adam [6] and RMSProp [14].

2.2.1 Variance Reduction. If the trajectory is quite long, chances are that our noisy gradient estimator is of high variance[2]. One method is to subtract a baseline function $b(s_t)$ from our estimator (5):

$$\sum_{t=0}^{T-1} \nabla_\theta \, \mathrm{E}[r_t|\pi_\theta] = \mathop{\mathrm{E}}_{\tau\sim P_\theta(\tau)}\left[\sum_{t=0}^{T-1}\left(\nabla_\theta \log \pi_\theta(a_t|s_t)\left(\sum_{t'=t}^{T-1} r_{t'}(\tau) - b(s_t)\right)\right)\right].$$

The equality holds for arbitrary function $b(s_t)$. See details in [10]. The quantity $\sum_{t'=t}^{T-1} r_{t'}(\tau) - b(s_t)$, which is used to scale the score function, can be regarded as the advantage of taking action a_t at step t. A near optimal baseline function is the state-value function[4]

$$V^\pi(s_t) = \mathrm{E}\left[\sum_{i=t}^{T-1} r_i | s_t, \pi\right].$$

This choice also reflects the idea of advantage in that the difference between actual rewards and expected rewards is naturally a good representation of the advantage function.

Another method can be utilized to reduce the variance is to add discount factor $0 < \gamma < 1$ to future rewards[3]. This reflects a bias-variance trade-off because adding γ decreases the variance but increases the bias. Then rewrite our estimator with such a state-value function as

$$\mathop{\mathrm{E}}_{\tau\sim P_\theta(\tau)}\left[\sum_{t=0}^{T-1}\left(\nabla_\theta \log \pi_\theta(a_t|s_t)\left(\sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'}(\tau) - V^{\pi,\gamma}(s_t)\right)\right)\right]$$

where

$$V^{\pi,\gamma}(s_t) = \mathrm{E}\left[\sum_{i=t}^{T-1} \gamma^{t'-t} r_i | s_t, \pi\right].$$

We denote $\sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'}(\tau) - V^{\pi,\gamma}(s_t)$ by $A(s_t, a_t)$ for simplicity.

The third method is to use multiple agents to go for independent trajectories and then take the average of their gradients to update the parameter[4].

All these three methods add up to asynchronous advantage actor-critic (A3C) method [9], which is also used by this paper.

3 RELATED WORK

For 360-video streaming, multi-layer video coding scheme is utilized to reduce the overheads of useless bandwidth[13]. Current works concentrate on using other assisted techniques[2] like PI controller to manage the interplay between layers or buffers[1][11].

Recently, the idea of using DRL for video streaming has been implemented in Pensieve[8]. It demonstrates Pensieve is close to optimality in the sense that without future bandwidth knowledge, it almost performs the best. Their promising results also indicate DRL embraces potential in 360-degree video streaming.

4 METHODS

4.1 Field of View (FoV) Prediction

We treat the FoV prediction problem as sequence prediction and propose sequence learning based prediction models. Most of related works focused on using the target user's own history to predict his/her future. In our work, we assume that the video server has stored the FoV trajectories of other viewers who have watched the same video and make use of the other viewers' trajectories to help the prediction of the target viewer's future trajectory.

We focus on long-term prediction and perform prediction at the second level, rather than at the frame level. We use the actual trajectory over each past second as the available information, and predict the mean and standard deviation of the trajectory over each future second. The client or server can decide on the video chunk coverage for a future second based on the predicted mean and variance of the FoV center.

Single LSTM is a popular solution in the related works such as [**?**] [**?**]. The future FoVs are predicted by unrolling the same LSTM. The underlying assumption that the past trajectories' distribution is the same as the futures' may not hold. Though FoV trajectories have a very short-term high auto-correlations [**?**], keep a longer memory may be advantageous for some cases. We adopt the sentence translation architecture [**?**] for our FoV trajectory prediction, as its encoder-decoder structure allows a higher flexibility for modeling the past and future distributions. Specifically, we use a LSTM to encode the past trajectory in time 0 to T-1, and use the last hidden state h_T and memory state c_T as the representation of the past. We then use another LSTM initialized by h_T and c_T and an initial input x_T, to generate the hidden/memory state. We further added an MLP mixing module to automatically learn the mixing weights of different users. The final prediction is obtained using one projection layer from the mixed hidden state.

4.1.1 FoV Accuracy Model. Based on the idea that if the accuracy of the FoV predictor is available to the streaming agent, then it might be able to incorporate that information to make a better decision. That is, suppose it already knows the enhancement chunk that the FoV predictor tracks would be of low accuracy, then chances are that it would rather take a base chunk this time and defer to fetch the enhancement chunk. Intuitively the next time the FoV predictor may give a more accurate prediction because as the enhancement buffer is consumed it predicts a nearer future.

This information can be approximated if we track the FoV predictor's performance in real time as the user watches the video. That is, once the streaming agent knows the predicted trajectory of the FoV predictor and also the true trajectory of the user vision, it can simply compare the trajectories to approximate the accuracy.

For our FoV accuracy model, we use the moving average of the past two decisions' accuracy. The reason why we use such a short moving average is that the FoV predictor's performance depends highly on the local video content. Very old accuracy probably expires.

4.2 Data Augmentation

In order to evaluated how RL based streaming model performs in WiGig network environment, several throughput traces are collected under different levels of interference. Each of the traces can be regared as one or more specific network status. As the training of a deep network requires a huge amount of data to support, especially for RL, training dataset should be diverse and aabundant enough to allow the RL agent to encounter all the possible

[2]To see this, consider random walk $y_0 = \epsilon_0; y_t = y_{t-1} + \epsilon_t$ where ϵ_i are Gaussian iid noise with variance σ^2. Then $\mathrm{Var}[\sum_{t=1}^{N} y_t] = N\sigma^2$.
[3]Briefly, $\mathrm{Var}[\gamma X] = \gamma^2 \mathrm{Var}[X]$.
[4]Suppose X_i, $i = 1, \ldots, N$ are iid, then $\mathrm{Var}\left[\frac{\sum_{i=1}^{N} X_i}{N}\right] = \frac{\mathrm{Var}[X]}{N}$.

states, we generate more synthetic WiGig throughput traces using a HMM-based approach.

First of all, based on the original data collected on the testbed, one specific HMM model is established to fit one of them. The number of hidden states in one HMM model depends on the performance of trace that more hiddens are utilized to represent more heterogeneous. The properties of each state and transition matrix among states are learned from the observed data by expectation maximization. Then, we manully design a transition matrix including probabilities of how HMM models are likely to transit to others or themselves. To make synthetic data more realistic, we integrat all the hidden states belonging to differernt HMM models into a heterogeneous one. This model can be considered as a comprehensive one that controls transition among all the state we collect.

4.3 Optimal Rate Allocation

When sending the video chunks from streaming servers to clients, the agent has to make decisions on which layer to fetch and which bitrate to choose. Given some assumptions, we can represent this problem as a constrained optimization problem. Typically, to maximize QoE, there is a closed-form solution with respect to which bitrate combinations we should select.

For a mathematical expression, we need to formalize the notations. Suppose base layer bitrate is R_b, enhancement layer bitrate is R_e, bandwidth is B. Denote the QoE by $Q(r)$, which is function of effective bitrate. Also, for effective usage of the bandwidth, the 360-degree video is splitted into tiles and only user vision covered tiles are sent. Hence at the client end, the video rendering rate is R_b/A_b for base layer, where A_b is the coverage area of a base chunk; R_e/A_e for enhancement layer, where A_e is the coverage area of an enhancement chunk.

4.3.1 Layered Coding. Layered coding assumes enhancement layer is a complement of base layer. Base layer is a must but enhancement layer is optional. So here we assume base chunks are always available. Given enhancement chunks arriving rate γ and FoV prediction accuracy α, as we want to select the bitrates that maximize the QoE, we can express it as a constrained optimization problem[11]

$$\max_{R_b,R_e} \alpha\gamma Q\left(\frac{R_e}{A_e}+\frac{R_b}{A_b}\right)+(1-\alpha\gamma)Q\left(\frac{R_b}{A_b}\right)$$
$$\text{subject to } R_b+R_e=B.$$

Suppose QoE takes a logrithm form and solve this problem we get optimal rate allocation

$$R_b=\frac{(1-\alpha\gamma)A_bB}{A_b-A_e},$$
$$R_e=B-\frac{(1-\alpha\gamma)A_bB}{A_b-A_e}.$$

4.3.2 Non-layered Coding. The non-layered coding scheme assumes that the base layer and the enhancement layer are necessarily independent. Unlike layered coding, enhancement chunk can be played by itself. But to get rid of black spot, here we still require base chunks are must available. Form a simpler constrained optimization problem

$$\max_{R_b,R_e} \alpha\gamma Q\left(\frac{R_e}{A_e}\right)+(1-\alpha\gamma)Q\left(\frac{R_b}{A_b}\right)$$
$$\text{subject to } R_b+R_e=B.$$

Suppose QoE takes a logrithm form and solve this problem we get optimal rate allocation

$$R_b=(1-\alpha\gamma)B,$$
$$R_e=\alpha\gamma B.$$

Given specific bandwidth condition, like 3G, 4G, or 5G, we calculate the range of the bandwidth and then use the above formulas to allocate bitrates for layers. For example, in some system we use non-layered coding, and bandwidth is 500Mbps. Set $\alpha\gamma=0.9$ and the resulting bitrates are $R_b=50$Mbps, $R_e=450$Mbps. It tells that on average, this setting will maximize the QoE.

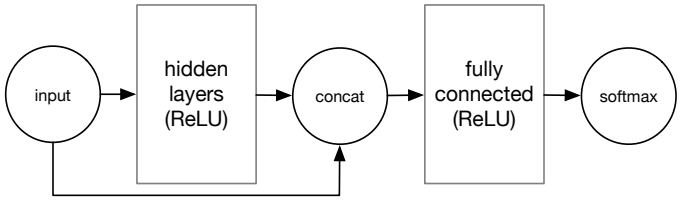

Figure 1: Model structure. The hidden layer is a complex of fully connected networks and 1-D convolution networks. The activation function is ReLU[3]. Information diffuses directly from input to the output of hidden layer through the residual connection. Policy distribution is computed from softmax node.

4.4 Streaming Agent

The streaming agent interacts with the environment. It takes actions to maximize the long-term rewards through the observations of the environmental and internal states. In the setting of 360-video streaming, the agent has to deal with a complex environment where not only bandwidth conditions vary a lot, base buffer and enhancement buffer are also closely correlated and need to be maintained simultaneously. Besides, under the presence of FoV predictor, the actions and rewards are also influenced by its prediction accuracy. All these elements are the considerations of an agent. For this task, the agents are not allowed to skip enhancement chunks or fetch a later chunk first, with only one exception when they cannot be downloaded in time.

4.4.1 Model Structure. As a decision unit, an agent is essentially a neural network taking internal and environmental states as input and predicts a corresponding action distribution $\pi_\theta(a|s)$. The structure is plotted in Figure 1. In the structure, the main part is a complex of fully connected networks, 1-D convolution networks[7], and residual connections[5]. The output is the probability distribution of actions. We simply arrange base layer bitrates and enhancement layer bitrates together in an increasing order to match the output of the softmax node.

The use of residual connections is justified in the experiment section. Here we only introduce it a little bit. In the setting of

layered coding scheme, we devised a special scenario for which we already know the optimal decisions. The pattern of decisions, for example, is to choose $R_b, R_e, R_b, R_e, \ldots$ iteratively. But it turns out it is impossible for the network without residual connections to learn the pattern. We notice from this repeated fashion that the next decision depends highly on the last decision, i.e., R_e follows R_b and R_b follows R_e for sure. The reason why it is hard to learn this pattern, we suspect, is the inability of a nonlinear network to model an identity mapping[5]. To counter this drawback, we add residual connections. Experiments support this idea.

The critic network follows the similar structure but with softmax node replaced with a single-layer fully connected network.

4.4.2 States Representation. To fully specify the environment, we use rich features. They are listed in table 1. Note that we use "BL", "EL" as abbreviations of "Base Layer" and "Enhancement Layer". All these features are scaled into the same magnitude. And all the convolution networks use filters with size 4. In the table, "dim" of the next chunk sizes is equal to the number of bitrates, which the agent is going to choose. Other input states with size greater than 1 are specified with previous history values. In our system, previous 8 values are specified. Most states' meanings are self-evident. For those not quite obvious, we provide an explanation here. First, end of video indicator is a zero-one scalar indicating if the buffer has reached the end of the video. Second, EL Rebuffering indicator is a zero-one scalar indicating the situation that video is being played without enhancement chunk, in which sense the enhancement layer is experiencing rebufferring. Third, FoV prediction accuracy is the module specified in section 4.1.1. It is a scalar that indicates the accuracy of the enhancement chunk to be fetched.

4.4.3 Reward Setting. For successful training of the model, an appropriate reward has to be set. Here we mainly capture three properties of a streaming agent.

First, we hope the streaming agent can make decisions that maximize the QoE. Second, the rebuffering time T_{rebuf} has to be made as little as possible. Third, we want the video to be played smoothly, not with huge fluctuation in bitrate, which is equivalent to minimizing the quality difference between adjacent chunks $|Q_k - Q_{k-1}|$. Sum them up, the reward for chunk k being played is

$$r_k = Q_k - w_1 T_{\text{rebuf}} - w_2 |Q_k - Q_{k-1}|,$$

where w_1 is the weight of rebuffering penalty and w_2 is the weight of smoothing penalty. For unit consistency, we simply choose w_1 equals the highest QoE and $w_2 = 1$.

Beside of these three general terms, we also need to consider some corner cases. In the above reward, it is not precise in that the enhancement chunk k may arrive in the middle of its playback so that Q_k is a mixed quality of base quality and enhanced quality. Thus, in order to capture this scenario precisely, we denote the actual enhancement chunk display time by T_E and suppose the chunk length is 1 second, then

$$Q_k = T_E Q(r_e) + (1 - T_E) Q(r_b).$$

In addition, the reward does not capture the playback turbulence in the above case, that is, when Q_k is a mixed quality. This turbulence penalty is expressed by the difference $Q(r_e) - Q(r_b)$, which is approximately 1. This turbulence has an effect on maintaining a longer length of enhancement buffer because the turbulence is likely to cause EL rebuffering, and less turbulence implies less EL rebuffering.

Finally, we exert a large penalty 1000 if a video layer has reached the end of the video but it still continues trying to fill in the layer by fetching non-exist chunks. We call this as blocking penalty because it acts like a barrier, giving a clear update signal to inform the agent of not doing this again. In total, our renewed reward is

$$\begin{aligned} r_k =& T_E Q(r_e) + (1 - T_E) Q(r_b) - w_1 T_{\text{rebuf}} - w_2 |Q_k - Q_{k-1}| \\ & - \mathbf{1}_{[\text{turb}]} - 1000 \times \mathbf{1}_{[\text{block}]}. \end{aligned}$$

4.4.4 Quality of Experience. As we described previously, the QoE is of logrithm form. For the training, we set the QoE as follows. We first use the highest effective bitrate as the normalizer Z to normalize all the other possible effective bitrates. Then we exert the logrithm to the normalized bitrates. Finally the bias b is added to the logrithm values. The bias aims to elevate the lowest QoE to a positive value 0.5 to provide a meaningful signal for the agent. Mathematically, for effective bitrate r, its QoE is

$$Q(r) = \log(r/Z) + b$$

where $Z = \frac{R_e^{\max}}{A_e} + \frac{R_b^{\max}}{A_b}$ and $b = -\log(r^{\min}/Z) + 0.5$.

4.5 Training Methods

As described in section 2.2, we make use of policy gradient learning algorithm to train our streaming agent. Among this branch, recently, A3C [9] stands out as an effective and efficient actor-critic algorithm. Our training strategy closely follows A3C.

Here, an actor aims to learn the policy, i.e., the action distribution $\pi(a|s)$. And a critic aims to approximate the state-value function $V^{\pi,\gamma}(s)$. Note that such a critic is devised only to decrease the gradient variance and help in faster training. Actor takes all the responsibility to make decisions during test.

Additionally, to encourage the agent to better explore the action space, entropy as a regularization term is added to the objective function. Thus we are maximizing both expected cumulative rewards and policy entropy. Larger entropy, intuitively, ensures the probability flatly spreads over the actions and the agent tends to take a rich combination of actions during streaming. Better exploration is critical for the agent to learn better decisions [8][15]. In practice, we exert an annealing entropy with a decreasing weight β. Concretely, the update rule becomes

$$\theta \leftarrow \theta + \delta \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) A(s_t, a_t) + \beta \nabla_\theta H(\pi_\theta(a|s))$$

where δ is the learning rate and β is the entropy decaying weight that decreases over time.

Moreover, in order to further reduce the variance in gradients and speed up the training, we take advantage of multiple independent agents to do exploration. In specific, for every iteration, the update signal is averaged among a bunch of gradients got from a set of independent agents. It is used to update a copy of parameters that is distributed to every agent at the beginning of each iteration.

Finally, some tricks are used for training the model. We would not train the model with the FoV predictor working concurrently. Instead, we first evaluate the accuracy of the FoV predictor on a

Table 1: States Representation

Input	Size	Network	Output Size
Last Video Bitrate	1	Fully Connected	128
BL Buffer Size History	8	Fully Connected	128
EL Buffer Size History	8	Fully Connected	128
BL End of Video Indicator	1	Fully Connected	128
EL End of Video Indicator	1	Fully Connected	128
BL Remaining Chunks	1	Fully Connected	128
EL Remaining Chunks	1	Fully Connected	128
EL Rebuffering Indicator	1	Fully Connected	128
FoV Prediction Accuracy for Next EL Chunk	1	Fully Connected	128
Video Chunk Size History	8	1-D Convolution	128*8
Download Time History	8	1-D Convolution	128*8
Next chunk sizes	dim	1-D Convolution	128*dim

set of user FoV traces. Then when training the model, the ground truth value of FoV prediction accuracy is used. This ensures the model to give full credit to FoV model accuracy, thus capturing the exact relation between FoV predictor accuracy and decision it makes. Only when testing the FoV accuracy model stated in section 4.1.1 is used. Besides, for an easier implementation and robustness, we do not allow enhancement buffer length longer than base buffer length. If that happens, we simply set the reward for that chunk to be zero.

5 EXPERIMENTS

In this section, we conduct experiments to demonstrate our motivations and show the properties of the streaming agent. Besides, we compare our model with another streaming method using PI controller[11]. Experiments show a positive improvement.

5.1 The Effects of Residual Connections

In order to learn about whether the potential of the fully connected neural network and convolution network is adequate for this double-layer streaming task. We devise a simple task which we already know the optimal solution and see if the streaming agent can find this solution.

In particular, we design a situation that the bandwidth is a constant value 500Mbps with utility rate 0.85 in the sense that we only use 85% of the bandwidth for streaming. We use layered coding scheme where we set $\alpha\gamma = 0.8$ and $A_e = 150 * 180, A_b = 360 * 180$. By the optimal rate allocation formula, the bitrate allocations are

$$R_b = 145\text{Mbps}, \quad R_e = 280\text{Mbps}.$$

These bitrates means that only using such bitrates to encode the video, under the bandwidth of 500Mbps, we can achieve the highest QoE. To see the optimal solution, we need another piece of block. We set the FoV prediction accuracy as a solid exponential curve: the accuracy decreases exponentially with the time period to be predicted ahead,

$$h(t) = \exp(-0.105t).$$

The parameter is set to have $h(1) = 0.9$, i.e., the accuracy of predicting the future FoV in one second is 0.9. It is equivalent to restrict the length of the enhancement buffer to be short, because longer buffer will result in lower FoV prediction accuracy, thus lower reward. This solid FoV accuracy curve, along with the turbulence penalty in the reward, result in a trade-off in the enhancement buffer length. Experiments show the length is maintained at 1 to 2 seconds.

To test if the agent is able to find the highest QoE, we add some interferences. We add other two base bitrates 75Mbps and 215Mbps for the network to distinguish. To sum up, the base bitrates involve 75Mbps, 145Mbps, and 215Mbps; the enhancement layer has a single optimal bitrate 280Mbps. We set chunk length is one second and initialization is one base chunk and one enhancement chunk available. Hence the optimal decisions are simply fetching 145Mbps, 280Mbps, 145Mbps, 280Mbps, ... because any other pattern of decision will incur penalties in rebuffering, smoothness, or low FoV prediction accuracy. From the optimal decisions, an iterative base chunk- enhancement chunk pattern appears. So we conjecture there is a high correlation between current decision and previous decision, and we employ such regularization as a residual connection in the network.

Based on our reward setting, the maximum total reward of fetching a 45-second video, in an ideal environment, is about 120. We trained both models (with or without residual connections) upto 70,000 iterations. However, without residual connections, the total reward can only achieve at most 94. On the contrary, the total reward of the model with residual connections can achieve as high as 107. See the decision patterns of the two models in figure 2.

In the figure, Y-axis has the indices of bitrates where 0 corresponds to the lowest bitrate in the base layer and 3 corresponds to the highest bitrate in the enhancement layer. It can be seen that an obvious approximate optimal decision pattern is made by the model with residual connections (bottom plot) except sporadic inconsistency, whereas the model without residual connections (upper plot) does not capture such patterns. Therefore, we conclude that residual connections provide good regularization for the training of the network and it only adds a small amount of computation.

Also note that in this task, we did not use the FoV accuracy input but make the model learn this information from the reward directly.

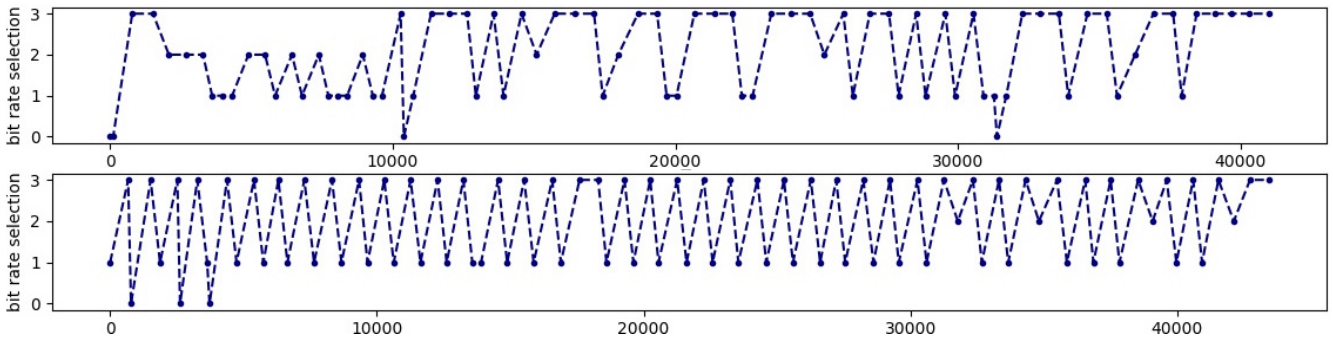

Figure 2: Comparison between models with or without residual connections. The upper plot is for the model without residual connections and the bottom plot is for the model with residual connections. Y-axis has the indices of bitrates where the correspondences are 0 - 75Mbps, 1 - 145Mbps, 2 - 215Mbps, 3 - 280Mbps. X-axis has milliseconds as units. Every point in the graph is a decision the streaming agent takes.

5.2 Optimality

In this section, we explore our proposed model's ability to make optimal decisions. In fact, in practical online setting, any model can never achieve the optimal decisions due to the absence of the future FoV and bandwidth knowledge. But in some cases that we already know the optimal solution, maybe we can use them for an illustration of the approximate optimality.

This time we train the network with practical bandwidth and test them on specific synthetic bandwidth for which we know the optimal solution as in the previous section. The bandwidth dynamics are the same as ones used in Pensieve[8]. But here we use re-normalization to re-scale them into bandwidths with averages between 500Mbps and 950Mbps. The re-normalization only scales up the bandwidth but preserves the dynamics. The model with residual connections is trained on 127 bandwidths of different dynamics. We also compute the optimal bandwidth allocation for bitrates. Using the formulas, we set

$$\begin{aligned}\mathbf{R}_b &= [85, 165] \\ \mathbf{R}_t &= [340, 490, 640],\end{aligned}$$

among which, 85Mbps and 340Mbps are a pair corresponding to bandwidth of 500Mbps; 165Mbps and 640Mbps are a pair corresponding to bandwidth of 950Mbps.

After training, we test the model on constant bandwidth of 500Mbps and 950Mbps, for which we hope the network could fetch (85Mbps, 340Mbps) and (165Mbps, 640Mbps) respectively. Encouragingly, the streaming agent does give the optimal solution on those bandwidth. In figure 3, it shows that the network can achieve approximately optimal solutions for these two synthetic cases.

Note that in this task, similarly to the previous one, FoV accuracy input is not used.

5.3 The Presence of FoV Accuracy

In this section, we empirically show that the availability of FoV accuracy information is beneficial for the streaming agent to make better decisions.

We follow the same setting in preivous section, but replace the solid FoV accuracy exponential model with the ground truth FoV accuracy and take it as an input state. We trained this model on the same bandwidth set and 39 collected user FoV traces. We tested both models on 142 different bandwidth and 1 left-out user FoV trace. The results are compared in a statistical way. In figure 4, we plot the cumulative density function (cdf) of rewards for two models. It shows that on most bandwidth, the model with information of FoV accuracy performs better. This confirms the importance of the FoV accuracy knowledge.

Besides, we can use a special case to show that the model does learn the FoV accuracy information and utilize it for streaming. We compose a synthetic FoV accuracy curve that gives flipping 1.0 and 0.0 every second, that is, outputs 1.0 at odd seconds and 0.0 at even seconds. This curve corresponds to a very weired FoV predictor that predicts exactly correct at odd seconds and does not work at all at even seconds. Intuitively, when the FoV accuracy is 1.0, the streaming agent should tend to fetch enhancement chunks; when the FoV accuracy is 0.0, the streaming agent should tend to fetch base chunks. This intuition is justified in our experiment illustrated in figure 5. As is shown, there is a positive correlation between FoV accuracy and which layer that is fetched. We use Pearson correlation coefficient to quantify this correlation. The coefficient turns out to be about 0.4, which supports a positive correlation. It also demonstrates the fact that the neural network is making decisions utilizing the FoV accuracy information.

We further compared our model with a traditional streaming method that uses PI controller[11]. In this task, training and testing bandwidth traces are generated using HMM specified in section 4.2. FoV prediction is achieved by the model in section 4.1. We let both systems perform a 45-second streaming task and then evaluate their performance with a uniform Quality of Experience (QoE) metric. For a video chunk k, its reward is computed as

$$r_k = 1.89 \log(R) - 1.518$$

where R is the displayed bitrate. The maximum reward for a 45-second video in an ideal environment is 465.2.

Hold other settings the same, we show the cdf of the rewards in figure 6. Our proposed DRL streaming agent consistently performs better: it gets an average score 433.7 while the baseline average score is 365.1.

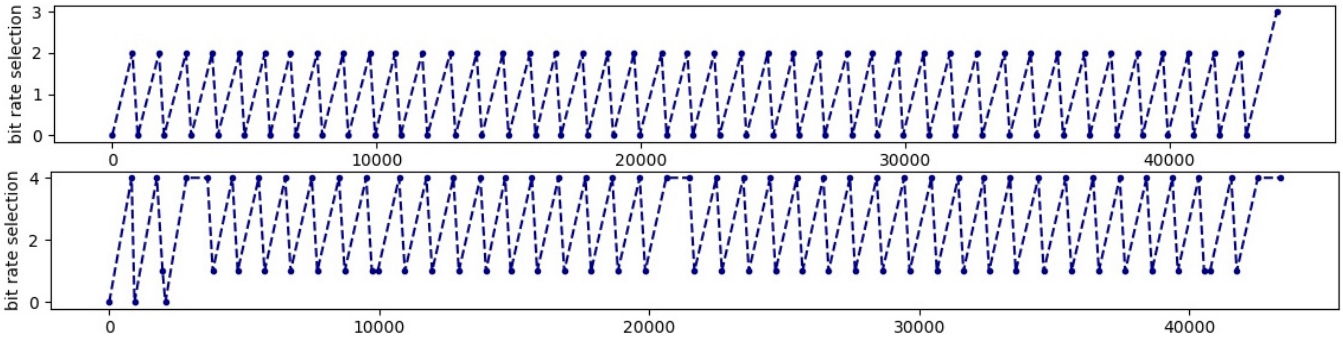

Figure 3: Optimal decisions for 500Mbps and 950Mbps bandwidth. The upper plot is for bandwidth of 500Mbps and the bottom plot is for the bandwidth of 950Mbps. Y-axis has the indices of bitrates where the correspondences are 0 - 85Mbps, 1 - 160Mbps, 2 - 340Mbps, 3 - 490Mbps, 4 - 640Mbps. X-axis has milliseconds as units. Every point in the graph is a decision the streaming agent takes.

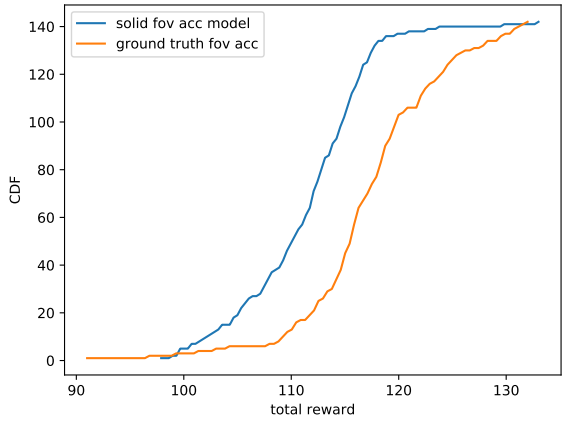

Figure 4: The effects of FoV accuracy information. Models are trained on different FoV accuracy signal and tested on the same test signal.

6 FUTURE WORK

In this section, we summarize the future work we need to do.

More options of the end positions of the residual connection need to be investigated.

More FoV accuracy models need to be investigated.

More forms of FoV accuracy input needs to be investigated. For example, we can also add FoV accuracy history information into the network.

More comprehensive optimality analysis needs to be made. For example, we only show that under non-layered coding scheme and for specific cases, the streaming agent can achieve optimal decisions. But we have not tested this widely for layered coding scheme, although we use it to justify our idea.

Experiments on different user FoV traces need to be done.

More fair experiments should be taken between our model and PI controller model. For example, we should use the same FoV predictor.

We believe new architecture of DRL model, like hierarchical model, is more suitable to this task. Different modules, for example, base layers, enhancement layers, and FoV predictor, fit into different factors and all of them can be trained jointly. This new direction can be further explored.

7 CONCLUSIONS

Deep reinforcement learning as a new technique is being used more and more in various domains. Video streaming is one of them[8]. In this paper, we particularly explore the application of DRL in 360-degree video streaming. The unique two-layer or multi-layer optimization problem posits a bunch of challenges for video streaming, making the system rather complex. We start from special cases with known optimal solutions, analyze their decision patterns, and use the insight from those patterns to regularize the model with residual connections. This regularization greatly improves the model's learning ability. In some cases, the approximate optimal decisions can be reached. Experiments illustrate this point. In addition, we exploit the FoV prediction accuracy of FoV predictors to further improve the streaming action. Experiment shows the network does learn this information. Finally, comparison between the DRL streaming agent and PI controller agent is conducted. It shows our proposed DRL streaming agent consistently performs better than PI controller method.

REFERENCES

[1] Fanyi Duanmu, Eymen Kurdoglu, S Amir Hosseini, Yong Liu, and Yao Wang. 2017. Prioritized buffer control in two-tier 360 video streaming. In *Proceedings of the Workshop on Virtual Reality and Augmented Reality Network*. ACM, 13–18.

[2] F. Duanmu, E. Kurdoglu, Y. Liu, and Y. Wang. 2017. View direction and bandwidth adaptive 360 degree video streaming using a two-tier system. In *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*. 1–4. https://doi.org/10.1109/ISCAS.2017.8050575

[3] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. 2011. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. 315–323.

[4] Evan Greensmith, Peter L Bartlett, and Jonathan Baxter. 2004. Variance reduction techniques for gradient estimates in reinforcement learning. *Journal of Machine Learning Research* 5, Nov (2004), 1471–1530.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

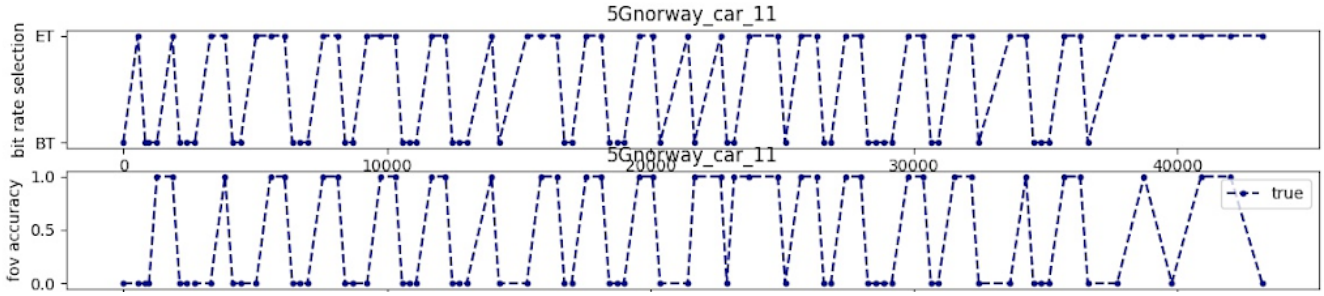

Figure 5: The effects of FoV accuracy on decisions. BT means Base tier/layer and ET means enhancement tier/layer. There is a positive correlation between FoV accuracy and which layer's chunks that are fetched.

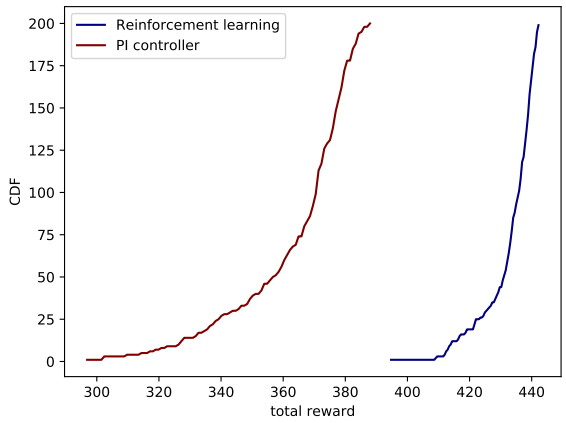

Figure 6: Comparison between PI Controller agent and DRL agent.

[6] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[7] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.

[8] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. 2017. Neural adaptive video streaming with pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 197–210.

[9] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*. 1928–1937.

[10] John Schulman. 2016. *Optimizing expectations: From deep reinforcement learning to stochastic computation graphs*. Ph.D. Dissertation. UC Berkeley.

[11] Liyang Sun, Fanyi Duanmu, Yong Liu, Yao Wang, Yinghua Ye, Hang Shi, and David Dai. 2018. Multi-path multi-tier 360-degree video streaming in 5G networks. In *Proceedings of the 9th ACM Multimedia Systems Conference*. ACM, 162–173.

[12] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. 2000. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*. 1057–1063.

[13] Afshin TaghaviNasrabadi, Anahita Mahzari, Joseph D Beshay, and Ravi Prakash. 2017. Adaptive 360-degree video streaming using layered video coding. In *Virtual Reality (VR), 2017 IEEE*. IEEE, 347–348.

[14] T. Tieleman and G. Hinton. 2012. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning.

[15] Yuxin Wu and Yuandong Tian. 2016. Training agent for first-person shooter game with actor-critic curriculum learning. (2016).