Learning From Deep Learning: a DRL-based Video Quality Aware Rate Control Approach

Tianchi Huang, Rui-Xiao Zhang, Ming Ma, Chao Zhou, Bing Yu, Lifeng Sun*

***Abstract*—Real-time video streaming is now one of the main applications in all network environments. Due to the fluctuation of throughput under various network conditions, how to choose a proper bitrate adaptively has become an upcoming and interesting issue. We notice that there exists a trade-off between sending bitrate and video quality, which motivates us to focus on how to get a balance between them. Motivated on this, we propose QARC (video Quality Aware Rate Control), a deep reinforcement learning (DRL) based rate control algorithm that aims to obtain a higher perceptual video quality with possibly lower sending rate and transmission latency. We evaluate QARC over a trace-driven emulation, outperforming existing approach with improvements in average video quality of 18% - 25% and decreases in average latency with 23% - 45%. In addition, to understand how QARC works, we continue to investigate the insight of QARC by reconstructing its network architecture to make it explainable. We receive useful insights in each of QARC's modules via inspecting their visualization attention heatmap, by which we can improve the model performance gradually. In general, through learning from deep learning, the advanced QARC achieves the same performance as the previously proposed scheme with an overhead reduction of about 90%.**

***Index Terms*—Rate Control, Deep Reinforcement Learning, XAI.**

I. Introduction

Recent years have witnessed a rapid increase in the requirement of real-time video streaming. People publish and watch live video streaming using different applications(e.g., Twitch, Kwai, Douyu) at any time, in anywhere, and especially, in any network environments. Due to the complicated environment and stochastic property in various network environments, transmitting video stream with high video bitrate and low latency has become the fundamental challenge in real-time live video streaming scenario. Most rate control approaches have been proposed to tackle the problem, e.g. loss-based approach (TFRC [5], RAP [6]), delay-based approach (Vegas [7], LEDBAT (Over UDP) [8]), and QoE-based approach (Google Congestion Control(GCC) [9], Rebera [10]). The same strategy of them is to select bitrate as high as possible with the permission of network condition. However, due to the inequality between high video quality and high bitrate, this strategy may cause a large waste of bandwidth resources. For example, as illustrated in Figure 1(a), if a video footage consists of darkness and few objects, a low bitrate may also provide a barely satisfactory perceptual video quality but can save large bandwidth resources.

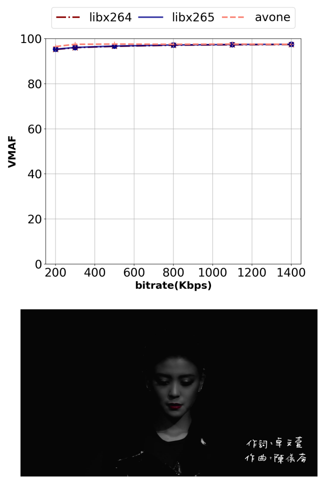
In previous work [11] we propose QARC(video Quality Aware Rate Control), a novel DRL-based rate control algorithm aiming to get high video quality and low latency. Due to that fixed rules fail to effectively handle the complicated scenarios caused by perplexing network conditions and various video content, we leverage DRL to select the future video bitrate, which can adjust itself automatically to the variety of its inputs. In details, QARC uses DRL method to train a neural network to select the bitrate for future video frames based on past time network status observed and historical video frames. However, if we directly import raw pictures as the inputs of state, the state space will cause "state explosion" [12]. To tackle these challenges, we meticulously divide this complexed RL model into two feasible and useful models: one is Video Quality Prediction Network (VQPN), which can predict future video quality via previous video frames; the other is Video Quality Reinforcement Learning (VQRL). VQRL uses A3C [13], a novel DRL method, to train the neural network. The state of the VQRL are past time network status observed and future video quality predicted by VQPN, and the action means the bitrate for the next video with high video quality and low latency. We design the training methodologies for those two neural networks respectively. To train VQPN, in addition to some general test video clips, we build up a dataset consisting of various types videos including movies, live-cast shows, and music videos. For training VQRL, we propose an offline acceleration network simulator to emulate real-world network environment with a trace-driven dataset. We then collect a corpus of network traces for the simulator with both packet-level traces and chunk-level public traces.

For evaluating QARC, we compare QARC with existing proposed approaches. Results of trace-driven emulation show that QARC outperforms with existing proposed approaches, with improvements in average video quality of 18% - 25% and decreases in average queuing delay of 23% - 45%. Besides that, by comparing the performance of QARC with the baseline which represents the offline optimal based on high bitrate and low latency over different network conditions and videos, we find that in all considered scenarios, despite a decrease in average video quality of only 4% - 9%, QARC saves the sending rate with 46% to 60% and reduces the average queuing delay of 40% to 50%.

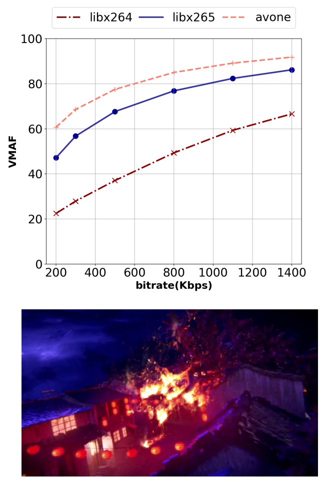
Previous work only focuses on designing high-performance neural networks as a "black box" without considering whether they are explainable or not. Yet, the ignorance of decisions or recommendations on neural networks may seriously constrain its further improvements. Motivated by the increasing development of explainable artificial intelligence(XAI) [14], [15],

* Corresponding Author

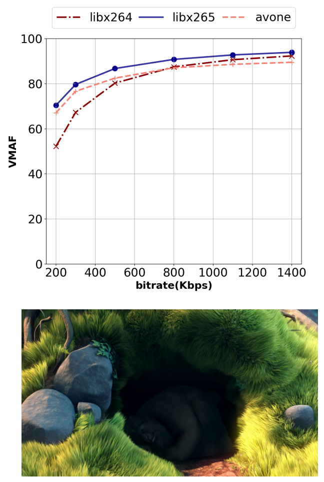
Manuscript received July 21, 2018; revised Oct. 26, 2018.



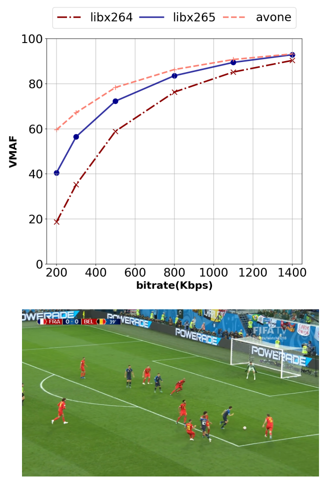
(a) “Be yourself” [1]: a music video clip with less video scene changes.



(b) “I’m not yours” [2]: a music video clip with dynamic and complicated video scene changes.



(c) “Big Buck Bunny” [3]: a comic video clip with both static and dynamic video scene changes.



(d) “World Cup Live” [4]: a soccer game video clip with both static and dynamic video scene.

Fig. 1. This group of figures shows our motivation: In the real-time live streaming scenario, high video bitrate is equaled to high video quality, however, in some circumstance, high video quality only requires a low bitrate.

[16], [17], in this work, we further explore the reasons for the superior performance of QARC, named as “learning from deep learning”. Our works consists of several steps. Firstly, on the basis of maintaining the performance, we reconstruct all the neural network modules in QARC, that involves:

- We design a novel hybrid attention-based CNN+GRU model to re-characterization VQPN. Results show that the proposed method reduces the network parameter sizes of 83% and improves the performance of 10% compared with the previous work.
- Inspired by [17], we reasonably replace the 1D-CNN layers to 2D-CNN for VQRL. Besides, we also use the global average pooling layer to take the place of the fully-connected layer. Through the results we find that the proposed method build a visual heatmap representation that explains the implicit attention of VQRL for each input of the state. Experimental results demonstrate that the proposed method generalizes well, with improvements in average reward of 3.6%.

We then leverage the attention heatmap generated by the explainable VQPN and VQRL to investigate the QARC’s logic. We realize that the essence of image feature extraction process of VQPN can be replaced by computing the average value of the grayscale image, and prove that the FFT features in the input of VQRL is actually an important factor. Based on “learning from deep learning”, we propose advanced QARC, obtaining the close performance as the previous approaches by reducing about 90% of the model size overhead.

As a result, our contributions are multi-fold.

- Considering to explain how does the QARC work, we reconstruct QARC’s modules to the explainable network architectures. To the best of our knowledge, this is the first attempt to investigate the insight of the AI-based rate control method in the video streaming scenario.
- The most difficult challenge of this paper is to redesign the QARC’s network architecture without reducing its performances. The proposed explainable QARC performs better than previous work, with the improvements of 10% and 3.6% respectively, which overfulfill the challenge.
- “Learning from deep learning” is effective. Many helpful and useful information can be found on all of the modules in QARC. Through “learning” from VQPN, we observe that extracting features by computing average value of the grayscale image for each video frame are able to reflect the essence. Meanwhile, through “leaning” from VQRL, we find that the features extracted by FFT play a vital role in determining the next bitrate.
- Considering previously mentioned ideas, we propose an advanced QARC scheme. The scheme achieves the similar performance comparing with previous QARC, with the reducing the overhead of about 90%.

The remainder of the paper is organized as follows: In Section 2, we introduce our motivation of this work. In Section 3, we explain our previous rate control method. Section 4 describes the explainable QARC, a DRL-based rate control approach. In Section 5, we explain the implementation details for each module in QARC, and then evaluate QARC with trace-driven emulation. Section 6 provides an overview of related work on rate control method, video quality metrics and reinforcement learning approaches. Section 7 makes some discussions about some key issues of QARC. Section 8 concludes the paper.

II. Motivation

In this section, we start by designing several experiments to answer three questions:

- With the enhancement of video encoding technology, what will the correlation change between video quality and video bitrate?
- Can rate control process be regarded as a Markov Decision Process? Why we use DRL to solve the rate control matters?

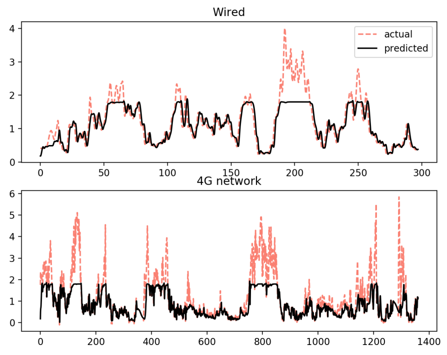


Fig. 2. On-line Training

- Can we precisely predict the fluctuation of the network without knowing the saturated bandwidth of the session via Deep Learning method?

A. *High Video Quality or High Video Bitrate*

To solve this, we establish a testbed to assess the video quality score of selected videos with the given encoding bitrate. The selected videos consist of three video clips, and each of them represents a video with static video scene (live-cast), a video with dynamic video scene (live concert), and a video with hybrid static video scene and dynamic video scene (MV) respectively.

In our experiment, we use Video Multi-Method Assessment Fusion(VMAF), a smart perceptual video quality assessment algorithm based on support vector machine(SVM) [18]. We compare the video quality score of each video in different encoders. In detail, we use three video encoders in our experiments including x264 [19], [20], x265 [21], and AV1 [22][1]. The first two encoders are popularly used nowadays, and the last one is the state-of-the-art video encoder proposed by Google.

As illustrated in Figure 1, comparing VMAF score of different encoders on different videos and encoded bitrates, the results show that as the encode bitrate increases, the rate of increase in video quality score decreases. In addition, the refinement of encoder technology does not eliminate this phenomenon. As a result, in the real-time live streaming scenario, if we blindly select the high bitrate, it will make the burden of the network transmission highly increase with little enhancement of video quality.

Inspired by this, we propose a novel sight which aims to optimize perceptual video quality rather than video bitrate during the entire video session.

[1]The arguments are aomenc -o test.webm src.y4m –codec=av1 –skip=0 -p 2 –good –cpu-used=1 –width=1280 –height=720 –target-bitrate=%d –lag-in-frames=25 –min-q=0 –max-q=63 –auto-alt-ref=1 –kf-max-dist=150 –kf-min-dist=0 –drop-frame=0 –static-thresh=0 –bias-pct=50 –minsection-pct=0 –maxsection-pct=2000 –arnr-maxframes=7 –arnr-strength=5 –sharpness=0 –undershoot-pct=100 –overshoot-pct=100 –frame-parallel=0 –tile-columns=0 –limit=150

B. *Estimate Future Network Status using Neural Network*

The second issue is focused on conventional network congestion control. In the real-time network scenario, by using a neural network, how to accurately estimate the future saturated bandwidth based on past network status observed is still a challenge. In this paper, we use machine learning approach to solve the problem, and in details, we use online learning to train a neural network model to predict the future network status.

Considering past k time-slots, we define $I_t = \{s, r, d\}$ as the input of neural network, where s is the sending rate of past k time-slots measured by the sender; r represents the receiving throughput collected by the receiver of past k time-slots, and d is the delay gradient computed by the receiver at that time-slot. In our experiment, we set $k = 5$. The output is a linear value described as the throughput of next time-slot $t + 1$, and in our problem, this value is equal to the available bandwidth. The model is mainly constructed as a 1D-convolutional network (1D-CNN). To train this model, We propose a network simulator which generates network status data via saturated traces, and more details can be seen in Section IV-C. In particular, sending rate is constrained in the range of $[0.01, 1.8]$ Mbps, which cannot reach the maximum size of available bandwidth.

$$\text{SMAPE} = \frac{100\%}{n} \sum_{t=1}^{n} \frac{|F_t - A_t|}{(|A_t| + |F_t|)/2}. \tag{1}$$

Figure 2 illustrates our results on real-world network datasets (Section. V-A). As shown, the model that is trained on the synthetic dataset is able to generalize across network conditions, and achieving SMAPE (Eq. 1) score within 11.1% of the model trained directly on the real-world networks including wired network and 4G network. These results suggest that, in practice, the neural network using 1d-CNN will have an ability to estimate future network status without measuring available bandwidth.

C. *Markov Decision Process*

As network status and video quality metrics can be regarded as a specific state, we follow the fundamental idea of Pensieve [23]. The lifetime of each rate control decision is modeled as a Markov Decision Process (MDP). The MDP is composed of 4-tuple $(State, Action, Transition, Reward)$, in which

- States: refer to the status of the target.
- Action: encodes actions can be performed to a target.
- Transaction function: describes the effect of each action in each state, which is defined as $\hat{State} \leftarrow State \times Action$.
- Reward function: symbols the immediate reward received after processing $Action$ to $State$.

State: In this work, we jointly consider the state as two subspaces:

$$State = State_v \cup State_n. \tag{2}$$

Where $State_v$ represents the video quality information, and $State_n$ reflects the current network status.

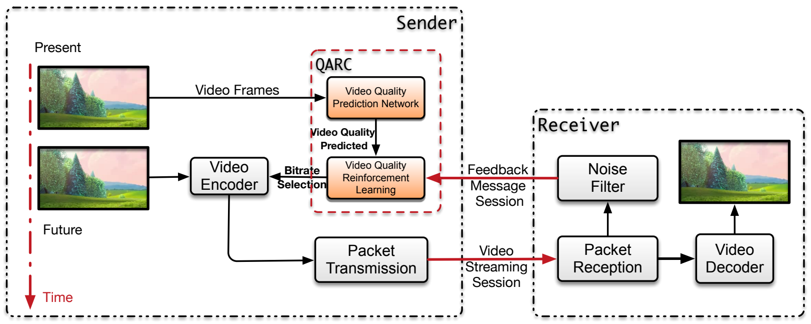


Fig. 3. The QARC's system architecture is composed of a video quality prediction model and a video quality reinforcement learning model.

Actions and Transaction Function: In our work, five possible transitions are designed between the states of each target, which is equivalent to five actions.

Reward function: In the MDP, there is no standard reward function because it needs to be changed from the target process. In this work, we denote the reward function as the quality of experience (QoE) metric. Details are provided in Section V-A.

Recent years have seen that various heuristics approaches (Rebera [10], GCC [9]) to tackle the rate control problem. However, these approaches suffer from a key issue: they all require tuning with great care and especially in the unexpected network conditions, unlike the assumption setup on, resulting in the failures of performs. Moreover, modeling a well-performed method which jointly refers to perceptual video quality and network status based on heuristics algorithm is quite a long and arduous task. We, therefore, solve an MDP with a DRL-generated approach which can automatically "learn" an optimized algorithm for different network conditions and video characteristics.

III. Previous QARC Overview

We start with explaining our previous proposed scheme [11] which is established on the conventional end-to-end transmission process for real-time video streaming. The system contains a sender and a receiver, and its transport protocol mainly consists of two channels: the streaming channel and the feedback message channel. At the beginning, the sender deploys a UDP socket channel to send the instant real-time video streaming packets $P = \{p_0, p_1, \cdots, p_k\}$, denoted as a packet train [24] to the receiver through the streaming channel. The receiver then feeds network status observed back to the sender through the feedback channel. Based on this information, the sender will select bitrate for next time period.

As shown in Figure 3, on the basis of conventional real-time video streaming system architecture, we propose QARC, a rate control approach which is placed on the sender side. Motivated by the unbalanced growth of video quality and video bitrate as described in Section II-A, we design a RL model to "learn" the correlation among the previous video frame, network status, and the best future bitrate. However, if we use raw pictures directly as its inputs, the state will cause "state explosion" [12]. Moreover, it will hard to train and validate in an allowable time. To overcome this, we meticulously partition the complexed RL model into two feasible and useful models, which involves:

- **Video Quality Prediction Network (VQPN)**, proposed by end-to-end deep learning method, which predicts the future video quality metrics based on historical video frames. Its architecture is described in Section V-B2;
- **Video Quality Reinforcement Learning (VQRL)**, which uses A3C, an effective actor-critic method that trains two neural networks to select bitrates for future video frames based on network status observations and the future video quality metrics predicted by VQPN.

IV. Explainable QARC's Mechanism

Previous work implemented a complete rate control approach based on DL and DRL, and performed well in various network conditions. However, due to the complex network architecture and unexplainable algorithms of deep learning, we still can't find out why QARC can finish such a complicated task. Ultimately, it will make it difficult for us to further enhance the performance of QARC. In order to better learn the how does the deep learning algorithm work, in this paper, we reconstruct QARC based on explainable artificial intelligence (XAI) and turned each of these modules into an interpretable machine learning architecture. In addition, a vital challenge for us is to preserve the performance of QARC as much as possible while constructing an explainable neural network.

In this section, we further describe the design and implementation of explainable QARC. We start by explaining the network architecture underlying explainable VQPN. We then describe the implementation details of explainable VQRL. Finally, we explain the offline network simulator and how it helps QARC to train the neural network models.

A. Video Quality Prediction Network(VQPN)

To help the RL model select a proper encoding bitrate for the next frame, we need to let the model "know" the relationship between the bitrate and corresponding video quality first. In other words, the model need to predict the corresponding video quality in different bitrate for future video frames. Meanwhile, forecasting contents of unseen queries for live video based on the improved recurrent neural network are the currently the "hotspot" of research. [25] This form of

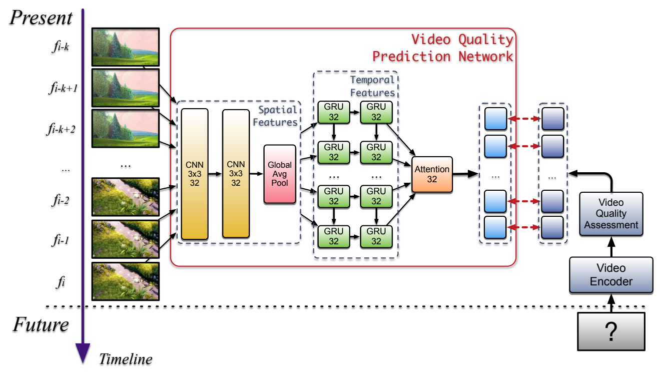


Fig. 4. VQPN Architecture Overview

prediction is quite challenging, because the subject video quality is closely related to the video itself. As shown in Figure 1, the video type, brightness, and objects number all have a great impact on the correlation between bitrate and VMAF. Motivated by the effectiveness of the neural network in a prediction of time sequence data, we design video quality prediction network(VQPN) to help the RL model understand the future frame's perceptual video quality. Figure 4 describes the VQPN's neural network architecture, which is mainly made up with a layer that extracts spatial information through Convolutional Neural Network (CNN), and another layer which capture temporarily features via Recurrent Neural Network (RNN). An attention layer is established after the RNN layer for demonstrating the implicit attention of the temporal information. Details are shown as follows.

Video Quality Metrics. We leverage "mean video quality metric" to describe the quality of the video over a period. For each raw video frame f_i in time-slot t, the video quality score $V_{f_i,bitrate}$ is computed by the raw video frames f and the bitrate at which the raw video frames f will be encoded, then the mean score $V_{t,bitrate}$ is defined as the average value of $V_{f,bitrate}$. In our study, we use mean VMAF [18], a video quality metric that is specifically formulated by Netflix to correlate strongly with subjective MOS scores to describe the video quality of video frames. In particular, we normalize the score into the distribution of the range from [0,1].

In this paper, we further list three different video quality metrics with their strengths and weaknesses in Section VIII-B, aiming to prove that in our scenario VMAF is the best method to evaluate video quality because the video is an image sequence with high temporal correlation, instead of a signal (PSNR) or a static picture (SSIM). Although calculating VMAF is quite inefficient, we still insist on using it to generate VQPN's datasets because our motivation is to bring the VQRL's results closer to the real quality of the video, and all the data generation process is finished in the offline environments. As a result, the overhead of VQPN are quite equal among that of three video quality schemes.

Input. VQPN takes state inputs $F_i = [f_{i-k}, f_{i-k+1}, \cdots, f_i]$ to its neural networks, in which f_i reflects the i-th sampled video frame.

Spatial information extraction. VQPN uses several 2D-convolutional layers to extract frame features, which can obtain the spatial information for each video frame in inputs F_i. Furthermore, we perform global average pooling layer (GAP) behind the last convolutional layer instead of fully-connected layer aiming to encourage VQPN to identify all discriminative parts of feature maps [17] and to make VQPN easier to interpret. Note that by removing the fully-connected layers, the number of neurons and the overhead of the network is all significantly decreased, even the performance is slightly improved compared with the previous work.

Capturing temporal features. Upon extracting frame features via CNN+GAP, VQPN leverages a double-layered recurrent layer [26] to further extract temporal characteristics of the video frames F_i in past k sequences. For illustrating the weights for each recurrent unit clearly, we present a deterministic "soft" attention [27], [28] layer behind the recurrent layer. In this work, we use the self-attentional model [29]. The idea is to calculate output V_{t+1} based on a context vector c_i with considering all the current hidden state h_i of the recurrent layer, and c_i is formulated as:

$$c_i = \sum_{i=1}^{k} \alpha_i h_i. \tag{3}$$

Where α_i is a variable-length alignment vector symbols the weight of each hidden state h_i, which can be obtained as follows:

$$\alpha_i = softmax(f_{att}(h_i)) \tag{4}$$

$$= \frac{exp(f_{att}(h_i))}{\sum_{j=1}^{k} exp(f_{att}(h_k))}. \tag{5}$$

Here, f_{att} is represented as an attention function that calculates an unnormalized alignment score. The attention function is defined as

$$f_{att}(h_i) = \mu_\theta^T \tanh(w_\theta h_i + b_\theta). \tag{6}$$

Where μ_θ, w_θ and b_θ are all learnable parameters. During the implementation, we complete the attention function as a single fully-connected layer neural network with $\tanh$ activation function. Hence, we have the ability to understand VQPN from beginning to end.

Output: The outputs of VQPN are the vector reflects the prediction of the video quality assessment in the next time slot $t+1$ encoded in several bitrates, denoted as V_{t+1}.

Loss function: We use mean square error(MSE) to describe the loss function. Besides that, we consider to add regulation to the loss function to decrease the probability of over fitting that on training set. To encourage the diversity in the attention vectors and to overcome the aforementioned shortcomings, we also add a new penalization formulated by Frobenius norm. Let $\hat{V_t}$ denote the real vector of video quality score of the video in time t. Therefore, the loss function can be written as (Eq. 7):

$$L_t(V;\theta) = \frac{1}{N} \sum |V_t - \hat{V_t}|^2 + \lambda_0 ||\theta||^2 + \lambda_1 ||\alpha\alpha^T - 1||_F^2 \tag{7}$$

where λ_0 and λ_2 are the regulation coefficient.

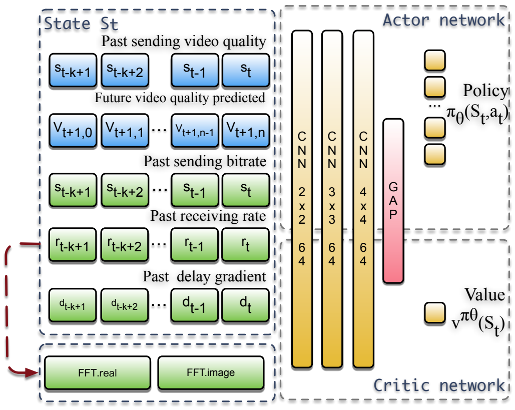


Fig. 5. The Actor-Critic algorithm that VQRL uses to generate sending bitrate selection policies

B. Video Quality Reinforcement Learning(VQRL)

In our study, we aim to let the neural network "learn" a video bitrate selection policy from observations instead of using preset rules in the form of fine-tuned heuristics. Specifically, our approach is based on RL. The sender, serving as an agent in RL problem, observes a set of metrics including future video quality and previous network status as the state. The neural network then selects the action as the output which denotes the video bitrate of next time-slot. Then the goal is to find the policy that maximizes the quality of experience (QoE) perceived by the user. In our scheme, QoE is influenced by video quality, latency, and smoothness.

As shown in Figure 5, We formulate "video quality first" real-time video streaming problem within A3C framework, named as video quality reinforcement learning (VQRL). Detailing our system components, which include:

State: We consider the metrics which can be obtained by both sender and receiver from feedback message session. VQRL's learning agent pushes the input state of time-slot t $s_t = \{p, v, s, r, d\}$ into neural network, where p means the past sending video quality, v is the future video quality predicted by VQPN, s is the video sending rate of the past k sequences which is equal to the throughput measurement from the uplink of the sender; r represents the receiving bitrate of past k sequences measured by the receiver; d is the delay gradient which is measured between a sender and receiver of the recent k sequences.

To better estimate the network condition in our scenario, we need precisely measure queuing delay of each packet. However, due to the clocks on both sides are unsynchronized, the measurements are unreliable. Like the scheme describes in [9], we also use delay gradient to solve the problem. Furthermore, we add the additional features into input which decomposed the receive rate sequence through FFT. The results that validate its improvement will be discussed in Section V-B2;

Action: The agent needs to take action when receiving the state, and the policy is the guide telling the agent which action will be selected in the RL problem. In general, the action space is discrete, and the output of the policy network is defined as a probability distribution: $f(s_t, a_t)$, meaning the probability of selection action a_t being in state s_t. In this paper, the action space contains the candidate of sending bitrate in the next time-slot.

In the traditional RL problem, the state space is small and can be represented as a tabular form, and there have been a lot of effective algorithms to solve this kind of problems, such as Q-learning and SARSA [30]. However, in our problem, the state space is fairly large, e.g., loss rate and received bit rate are continuous numbers, so it is impossible to store the state in a tabular form. To tackle this barrier, we use a neural network [31] to represent the policy, and the weights of the neural network, we use θ in this paper, are called the policy parameters. In recent researches, the technique of combining neural network and RL is widely used to solve large-state-space RL problems [32], [23] and shows its exceptional power;

Reward: As is mentioned in Section II-C, our reward is to maximum the Quality of experience (QoE) perceived by the user. In our scheme, QoE is influenced by video quality, latency, and smoothness. Details will be described in Section V-A.

Network Architecture Representation: The output of the actor network is a 5-dim vector with softmax activation, and the final output of the critic network is a linear value without activation function. In particular, they use the same network architecture which consists of two 2-D convolutional layers with 64 filters. Motivated by the key idea of [17], just before the last output layer of the actor network and the critic network, we also perform a global average pooling layer instead of a fully-connected layer. Our aim is to understand the weights for each input in the state. The implementation details will be found in Section V-B2.

Optimization: In the RL problem, after taking a specific action in state s_t, the agent will get a corresponding reward , and the goal for the RL agent is to find the best action in each state which can maximize the accumulated reward r_t and as a result, the policy should be changed in the direction of achieving this goal. In this paper, we use A3C [13] as the fundamental algorithm of our system, and in this algorithm, policy training is done by performing policy gradient algorithm.

The key thought of the policy gradient algorithm is to change the parameter in the direction of increasing the ac-

Algorithm 1 Overall Training Procedure

```
Require: The network simulator $\mathcal{S}$ to estimate the latency with given trace and network status; Pre-train the VQPN model $\mathcal{P}$ to predict the perceptual video quality.
1: Initialize the parameters of the policy network of VQRL model $\mathcal{L}_\theta$ with random weights.
2: repeat
3:     $f, n \leftarrow State$;        ▷ Get current frame and network status.
4:     $action \leftarrow \mathcal{L}_\theta(\mathcal{P}(f), n)$;
5:     $reward \leftarrow \mathcal{S}(action, n)$;
6:     Optimize $\mathcal{L}_\theta(reward)$;
7: until Converged
```

cumulated reward. The gradient direction is the direction in which a function increases. The gradient of the accumulated reward with respect to policy parameter θ can be written as:

$$\nabla E_{\pi_\theta}[\sum_{t=0}^{\infty} \gamma^t r_t] = E_{\pi_\theta}[\nabla_\theta log_{\pi_\theta}(s,a) A^{\pi_\theta}(s,a)] \quad (8)$$

and we can use:$E_\theta[\nabla_\theta log \pi_\theta(s,a) A^{\pi_\theta}(s,a)]$ as its unbiased form, where $A(s_t, a_t)$is called the advantage of action a_t in state s_t which satisfies the following equality:

$$A(a_t, s_t) = Q(a_t, s_t) - V(s_t) \quad (9)$$

where $V(s_t)$ is the estimate of the value function of state s_t and $Q(a_t, s_t)$ is the value of taking certain action at in state s_t, and it can also be written as:

$$Q(a_t, s_t) = r_t + \gamma V(s_{t+1}|\theta_{t+1}) \quad (10)$$

Thus, policy parameter will be updated as:

$$\theta \leftarrow \theta + \alpha \sum_t \nabla_\theta log \pi_\theta(s_t, a_t) A(s_t, a_t) \quad (11)$$

in which the parameter α represents the learning rate. To calculate $A(s_t, a_t)$, we need to have the $V(s_t)$ first, and we can estimate it in the value network. The value network aims to give a reasonable estimate of the actual value of the expected accumulated reward of state s_t, written as $V(s_t|\theta_v)$

Continuing the same line of thought, value network also uses neural network to represent the large state space. In this paper, we use n-step Q-learning to update the network parameter [13], and for each time, the error between estimation and true value can be represented as Eq.12:

$$Err_t = (r_t + \gamma V(s_{t+1}|\theta_v) - V(s_t|\theta_v))^2 \quad (12)$$

where $V(s_t|\theta_v)$ is the estimate of $V(s_t)$, and to reduce the Err_t, the direction of changing parameter θ_v is the negative gradient of it, and in A3C, the gradient will be added up with respect to t, so the value network will be updated as $\theta_v \leftarrow \theta_v - \sum_t \nabla_{\theta_v} Err_t$, where α is the learning rate. Inspired by [23], [13], we also add the entropy of policy in the object of policy network, which can effectively discourage converging to suboptimal policies. See more details in [13]. So the update of the theta will be rewritten as:

$$\theta \leftarrow \theta + \alpha \sum_t \nabla log_{\pi_\theta}(s_t, a_t) A(s_t, a_t) + \beta \nabla_\theta H(\pi_\theta(\cdot|s_t)) \quad (13)$$

where β is also a hyper-parameter that is set to decay from 1.0 to 0.15 over 10^5 iterations for controlling the probability of exploration. After convergence, the value network will be abandoned, and we only use policy network to make decisions.

Training: The complete training process is described in Algorithm 1. The training system is made up with a network simulator $\mathcal{S}$, a pre-trained VQPN model $\mathcal{P}$ and a VQRL model $\mathcal{L}$.

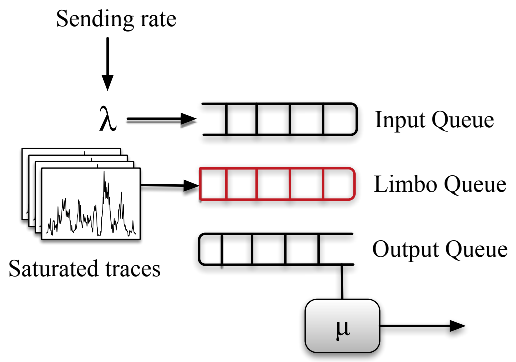


Fig. 6. The working principle of the network simulator.

Multiple training: To accelerate the training process, as suggested by [13], we modify VQRL's training in the single agent as training in multi-agents. Multi-agents training consists of two parts, a central agent and a group of forwarding propagation agents. The forward propagation agents only decide with both policy and critic via state inputs and neural network model received by the central agent for each step, then it sends the n-dim vector containing $\{state, action, reward\}$ to the central agent. The central agent uses the actor-critic algorithm to compute gradient and then updates its neural network model. Finally, the central agent pushes the newest model to each forward propagation agent. Note that this can happen asynchronously among all agents, for instance, there is no locking between agents. By default, VQRL with multiple training uses 8 forward propagation agents and 1 central agent;

C. Training with Network Simulator

To train VQRL, we first consider to train our neural network model in real-world network conditions, e.g., deploying the model on the edge server. With the increasing number of session, the model will finally converge. However, training the model online is hard to converge because RL training should meet almost all network status as the state. We then decide to train the model in simulated offline networks. Hence, we are facing new challenge: How to design a fast-forward network simulator which can precisely compute the latency with given saturated trace and sending rate?

We discuss to compute one-way delay, a metric that is widely used in congestion control algorithm. One-way delay consists of transmission delay, queuing delay and random noise. Predicting one-way delay requires detailed information for each router on the channel link between a sender and receiver. However, for each router on the session, there is always a special strategy with its own status and routing algorithms that we cannot collect. As a result, we can't estimate the accurate delay metrics in the offline environment with the parameters collected through our protocol.

Thus, we use queuing delay gradient rather than one-way delay to train our model. Delay gradient is defined as the difference on delay measurement on both side. [33] It is also proposed to measure the latency in unsynchronized clock environment. Delay gradient $q(t_i)$ is computed as follows[9]

(Eq. 14) , which is actually used to describe the variety of the queuing delay. If the network is congested, its delay gradient will be greater than zero, vice versa. If the delay gradient equals to zero, we can only infer that network is not congested.

$$q(t_i) = \frac{1}{N}\sum_{i=1}^{N}[(t_i - t_{i-1}) - (T_i - T_{i-1})] \tag{14}$$

Where T_i is the time at which the i-th packet has been sent, and t_i is the time at which the i-th packet has been received, and N represents the packet count in a period.

Our simulator should emulate the process of the packets coming and leaving in different network conditions, and keep track of the timestamps, by which we are able to obtain the corresponding queuing delay gradient. Inspired by Cellsim [34] and Mahimahi [35], we use saturated network trace to generate queuing delay data. Seen in Figure 6, assuming the distribution of packets arrival and leave fits closely to the Poisson process [34], we use sending bitrate and bandwidth in saturated network traces as the arriving rate λ and leaving rate μ, respectively.

In detail, we regard the channel link between the sender and the receiver as a single device. The device is comprised of three queues, namely "input", "output" and "limbo". An algorithm is designed to release packets from each queue corresponding to the network trace datasets in different levels. Each element in the queue is the opportunity of packet-delivery, which means, the time(in milliseconds) at which an MTU-sized packet will be delivered. For each period $(t_k, t_{k+1}]$, the offline-network simulator compares the timestamp of the front packet in "input" with that in "limbo". If timestamp in "input" queue is bigger than or equal to the one in "limbo", the simulator will push the front packet into the bottom of the "output" queue, and then computes the delay gradient between the two packets. On the contrast, the delivery opportunities will be wasted if the timestamp in "input" queue is smaller than that one in "limbo" queue. The simulator only pops the front packet in "limbo" queue. By this process, the simulator returns mean self-inflicted delay and total bytes that the sum of "output" queue every interval $t_{k+1} - t_k$.

V. Evaluation

A. Datasets and Metrics

Video dataset: To train and test VQPN, we use two video datasets.

- **VideoSet:** a large-scale compressed video quality dataset based on JND measurement. [36]
- **Self-collected video datasets:** a video quality dataset involves live-casts, music-videos, and some short movies. For each video in datasets, we measured its VMAF with the bitrate range [300kbps, 500kbps, 800kbps, 1100kbps, 1400kbps], and the reference resolution is configured as 800×480, which is the same size as default resolution observed by the receiver during the real-time live streaming. We generate the VMAF video datasets using both x264 [19] and x265 [21] encoder.

Network traces: To train and evaluate VQRL, the first thing we must do is to generate saturated network trace datasets. However, these types of network traces are hard to be recorded, even public datasets are extremely limited. For example, Cellsim only provides a small number of saturated network traces which describe the cellular network conditions instead of all network environments, which hardly afford us to make our neural network converge. Thus, we consider to collect datasets in three ways:

- **Packet-level network traces:** We use a proprietary dataset of packet-level live-cast session status from all platform's APPs of Kwai collected in January 2018. [2] The dataset, recorded as packet train, consists of over 14 million sessions from 47,000 users covering 50 thousand unique sessions over three days in January 2018. For each session, it is composed by packet size, packet send time and packet receive time. Based on raw data collected, we propose measuring the available bandwidth ABW/n of the whole link, where the available sample bandwidth is obtained by the packet train which is received in the receiving side in period n. Motivated by the one-way-delay estimation method in Ledbat [8], we generate 2,300 real network traces from packet-level network datasets.
- **Chunk-level network traces:** We also collect hybrid network traces datasets which consist of different network datasets, such as FCC [37] and Norway [38]. The FCC dataset is a broadband dataset, and Norway dataset is mainly collected in 3G/HSDPA environment. In short, we generate 1,000 network traces from the datasets.
- **Synthetic network traces:**We generate a synthetic dataset using a Markovian model where each state represented an average throughput in the aforementioned range.[23] Thus, we create a dataset in over 500 traces which can cover a board set of network conditions.

QoE metrics: For the better result, we consider designing Quality of Experience (QoE) metric based on previous scheme. In the recent research, QoE metrics are evaluated as a method with 4 essential factors: bitrate received, loss ratio, latency, and delay gradient, with no considering video quality metric. Still, in this paper, after rethinking the correspondence between video quality and video bitrate, we redefine the QoE metric as Eq. 15 for a live video with N time-slots.

$$QoE = \sum_{n=1}^{N}(V_n - \alpha B_n - \beta D_n) - \gamma\sum_{n=1}^{N-1}|V_n - V_{n-1}| \tag{15}$$

Where V_n denotes the video quality of time n, B_n is the video bitrate that the sender selects, and D_n represents the delay gradient measured by the receiver. The final term comprises the smoothness of video quality. Coefficient α, β and γ are the weight to describe their aggressiveness.

B. Implementation

We now introduced the implementation of QARC[3]. In this section, we decide the best hyper-parameters and explain the implementation of VQPN and VQRL respectively.

[2]Kwai is a leading platform in China which has over 700 million users worldwide, and millions of original videos are published on it every day.

[3]QARC are now available on https://github.com/godka/qarc

Time-slot t: In this paper, we set time-slot t to 1s.

1) VQPN's Implementation: The introduced VQPN helps VQRL predict future video quality, but we have not yet studied how well the model performs and how to set the hyper-parameters. Two experiments are designed to answer these two questions.

Experiment 1: We compare VQPN to the following learning methods which collectively represent the conventional prediction method:

1) Artificial neural networks (ANNs): uses a single layer with 64-dims neurons to predict future video quality;
2) Fully-connected neural networks (FC): uses 3 traditional fully-connected layers, and each layer is sized $\{128, 64, 64\}$ respectively;
3) Recurrent Neural Network (RNN): uses a conventional single-layered recurrent neural network with 128-dims hidden units;
4) Gated Recurrent Unit (GRU): uses an advanced double-layered recurrent neural network with the same hidden units as the one in VQPN;
5) Leverage specific input features including Spatial perceptual Information (SI) and Temporal Information (TI): as described in [39], SI is a measure that based on the *Sobel* filter, which generally indicates the amount of spatial details of a picture. Its process can be represented in equation form as Eq. 16, which means the maximum value (max_{time}) of the standard deviation (std_{space}) over the pixels for each frame that is filtered with the *Sobel* filter (Sobel (F_n)).
$$\text{SI} = max_{time}\{std_{space}[\text{Sobel } (F_n)]\}. \quad (16)$$
TI, a measure that generally indicates the amount of temporal changes of a video sequence, is computed as Eq. 17, in which $F_n(i,j)$ is the pixel at the i-th row and j-th column of n-th frame in time.
$$\text{TI} = max_{time}\{std_{space}[F_n(i,j) - F_{n-1}(i,j)]\}. \quad (17)$$
6) VQPN(previous work): Previously proposed VQPN architecture passes $t = 5$ past time video, and it samples 5 frames for each time, totally $k = 25$ previous frames as input to the neural network architecture, and each size of the frame is defined as width=64,height=36 in 3 channels. For each video frame in input frames then extract features in 128-dimension vector from a feature extraction layer respectively. The feature extraction layer is constructed with 5 layers, a convolution layer with 64 filters, each of size 5 with stride 1, an average pooling layer with size 3×3, an another convolution layer with 64 filters, each of size 3 with stride 1, also, an max pooling layer with size 2×2. The feature extraction layer then passes the features into a hidden layer with 64 neurons. A recurrent network is designed to estimate future video quality. VQPN passes k = 25 feature maps to two gated recurrent unit(GRU) layers with 64 hidden units. A hidden layer is then connected to the hidden output of the last GRU layer. The finite architecture of previous VQPN is defined as $Conv1a_{128} \rightarrow MaxP_{3\times3} \rightarrow Conv1b_{128} \rightarrow MaxP_{2\times2} \rightarrow Fc \rightarrow GRU2a_{64} \rightarrow GRU2b_{64} \rightarrow Fc4$.

TABLE I
COMPARING PERFORMANCE (RMSE%) OF VQPN WITH DIFFERENT REGRESSION ALGORITHMS INCLUDING ANN, FC, RNN, GRU AND SIMPLE LINEAR REGRESSION OF BOTH SPATIAL INFORMATION AND TEMPORAL INFORMATION.

Method	RMSE	Improvement(%)
ANN	0.181	-
FC(Baseline)	0.068	-
RNN	0.056	17.65
GRU	0.060	11.76
SI,TI	0.064	5.88
VQPN(previous work)	0.047	30.88
VQPN(our work)	**0.043**	**36.76**

TABLE II
COMPARING THE PERFORMANCE (RMSE%) AND THE OVERHEAD OF VQPN WITH DIFFERENT CHANNEL / HIDDEN UNITS. RESULTS ARE COLLECTED UNDER LEARNING RATE=1E-4.

Channel / Hidden Units	RMSE	Model Size(MB)
2	0.054	0.022
4	0.055	0.086
8	0.059	0.33
16	0.046	1.4
32	**0.043**	**5.3**
64	0.052	21.0
128	0.052	84.0

The comparison of Root Mean Square Error (RMSE) with VQPN against other regression algorithms is shown in Table I. RMSE is defined as Eq. 18, in which y_t represents the actual value and $\hat{y}_t$ is the forecast value. As expected, the obtained results indicate that VQPN succeeds in improving the prediction quality, with improvements of 36.76%. Note that we compare the performance of our approach and our previous work on the same test datasets. We observe that our approach still generalizes well, with improvements in RMSE of 8.5%.
$$\text{RMSE} = \sqrt{\frac{\sum_{t=1}^{n}(\hat{y}_t - y_t)^2}{n}}. \quad (18)$$

Experiment 2: Table II shows our results with different settings of channel number / hidden units. Results are summarized as RMSE metric. Empirically, channel number / hidden units = 32 yields the best performance. Meanwhile, its overhead and the model size can be accepted and deployed in the real-world scenarios.

Generalization: To sum up, like previous work, the explainable VQPN also passes $t = 5s$ past time video, and it samples 5 frames for each time, totally $k = 25$ previous frames as input to the neural network architecture, and each size of the frame is defined as 64×36 with 3 channels. The features of the input frames are extracted into a 32-dimension vector via

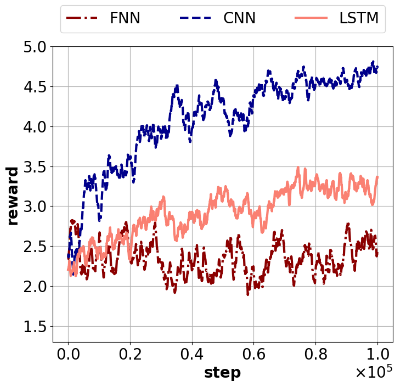


Fig. 7. Comparing the performance as different neural network model including CNN, FNN, and GRU.

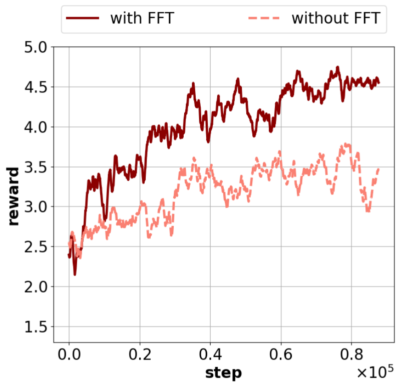


Fig. 8. Comparing VQRL which uses FFT with the one without using it.

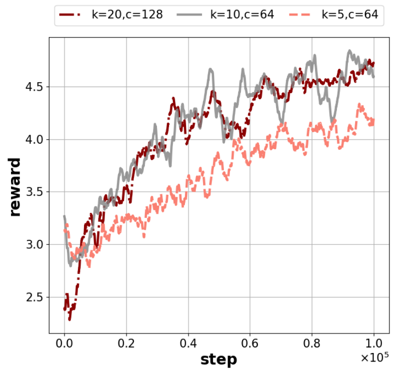


Fig. 9. Sweeping sequence length and number of filters in VQRL's neural network architecture.

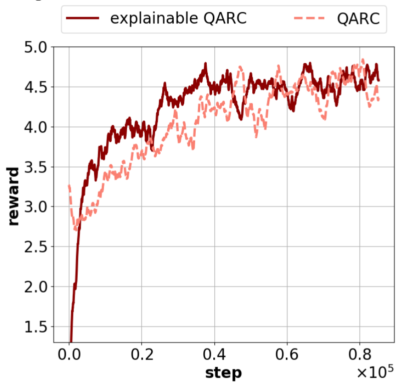


Fig. 10. Comparing the performance as different neural network model including QARC and previously proposed QARC.

spatial information extraction layer. The spatial information extraction layer is constructed with 2 convolution layers, both of which consist of 32 filters, size 3 with stride 1. There is no max pooling layer in the spatial information extraction layer. Finally, the feature map is passed into a global average pooling layer.

After extracted the spatial information for each video frame, a recurrent network is designed to estimate future video quality. VQPN passes k = 25 feature maps to a gated recurrent unit(GRU) layer with 32 hidden units, then the states of that layer are passed to another GRU layer with the same hidden units. A self-attention layer is performed behind the GRU layer. Finally, VQPN uses the final output as a 5-dimension vector, and for each value in the vector represents the video quality score of video bitrate $\{300, 500, 800, 1100, 1400\}$ kbps. In short, the finite architecture of previous VQPN is defined as $Conv1a_{32} \rightarrow Conv1b_{32} \rightarrow GAP \rightarrow GRU2a_{32} \rightarrow GRU2b_{32} \rightarrow Attention3_{32} \rightarrow Fc4$.

During the training process, we use Adam gradient optimizer [40] to optimize VQPN with learning rate $\alpha = 10^{-4}$. In this work, we use TensorFlow [41] to implement this architecture, in particular, we leveraged the TFLearn deep learning library's TensorFlow API to declare VQPN.

2) VQRL's Implementation: In this subsection, we describe how to choose the best neural network model of VQRL via three solid experiments.

Experiment1: For evaluating the performance of VQRL, we design three different models which represent the popular and efficient deep learning methods recently:

1) FNN(Feed-forward Neural Network): leverages 3 fully-connected layers where the neuron for each layer are {64,128,64} respectively.($Fc1a_{64} \rightarrow Fc1b_{128} \rightarrow Fc1c_{64}$)
2) LSTM(Long-Short Term Memory): uses double-layered LSTM layers in which the number of hidden units is 64, to simplify: $LSTM1a_{64} \rightarrow LSTM1b_{64}$.
3) VQRL(Previously proposed method): uses several 1D-convolution layer with 64 filters and one fully connected layer with 64 neurons to describe it.

We set sequence length $k = 5$,We use the QoE metric with $\alpha = 0.2$, $\beta = 1.0$ and $\gamma = 1.0$ as the baseline reward. As illustrated in Figure 7, the previously proposed VQRL model increases the average QoE by about 39% compared with the LSTM model and about 83% compared with the FNN model.

Experiment2: To validate the importance of adding FFT feature into inputs, we set up two CNN models, and one of them is established with FFT feature. We set sequence length $k = 20$ with the same environment as the first experiment. Results are shown in Figure 8, which implies that the CNN model with using FFT feature can provide a high reward with the improvement of about 29% compared with the CNN model without using FFT feature.

Experiment3: We now start to investigate how CNN's parameters inflect output results. In our experiment, the different parameters are set as $\{k = 5, c = 64\}$, $\{k = 10, c = 64\}$ and $\{k = 20, c = 128\}$, in which k corresponds to the input sequence length and c is the CNN channel size. As shown in Figure 9, with the increase of k and c, the performance increases. However, when we choose parameter $\{ k = 20, c = 128\}$, the average QoE only increases 1% compared with parameter $\{k = 10, c = 128\}$, so in consideration of calculation complexity, we finally choose $\{k = 10, c = 64\}$. Additionally, the action space is configured as 5, which is same as the output of VQPN. During the training process, we use Adam gradient optimizer to optimize it, and the learning rate for the actor and critic is set as 6.75×10^{-4} and 10^{-3}, respectively.

Experiment4: For evaluating the performance of explainable VQRL architecture, we experiment with the previous proposed VQRL and the explainable VQRL, the results are reported in Section IV-B and Experiment 3 respectively. Experimental results is shown in Figure 10. We observe that the explainable QARC generalizes better than QARC with improvements in average reward of 3.6%.

Training time: To measure the performance limitation of predicting future video quality, we profile VQPN's training process. To know when the network converges, we use early stopping method to train the neural network. Totally, training VQPN requires approximately an hour on a single GPU GTX-1080Ti. Furthermore, for measuring the overhead of the neural network of VQRL, we also introduce the training process for it. To train this, we use 8 agents to update the parameters of the central agent in parallel. The neural network will converge in 22 hours, or less than 5 hours using 20 agents[4].

C. Experiments and Results

In this section, we establish a real-time video streaming system to experimentally evaluate QARC via trace-driven emulation. Our results answer the following questions:

1) Comparing QARC with previously proposed approaches in different video clips, is QARC the best scheme?
2) Compared with the baseline algorithm based on high video bitrate and low latency, how much improvement does QARC gain on the results?

Experiment 1: QARC vs. Heuristic Methods. In this experiment, we evaluate QARC with existing proposed heuristic methods on several network traces which represent various network conditions by using trace-driven emulation. After running the trace for each approach, we collect the average queuing delay, average video quality and average sending rate from the receiver. We compare their performance via different video clips which are made up with a "less scene change" video clip [1], a "frequent scene change" video clip [2] and a video sample in both "static" and "dynamic" scene changes [3]. In this experiment, QARC is applied to compare the performance with Google Hangout, a famous video conference app, Compound TCP[42], and Vegas [7]. As illustrated in Figure 11, one of the results show that QARC outperforms with existing proposed approaches, with

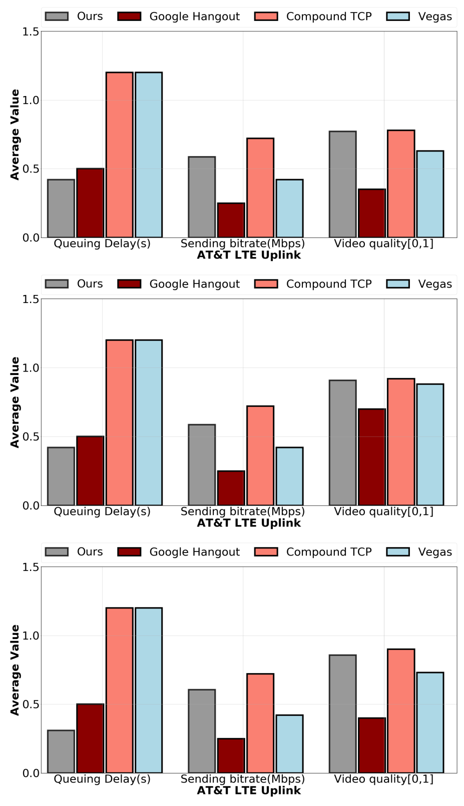


Fig. 11. Comparing QARC with previous proposed approaches on the 4G network environments: The QoE of QARC is considered as $\alpha = 0.2$, $\beta = 10.0$,and $\gamma = 1.0$. After testing three video clips, results are shown as average queuing delay, average sending rates, and average video quality.

improvements in average video quality of 18% - 25% and decreases in average queuing delay of 23% - 45%. Especially, we obtain that QARC also saves the sending rate, and also performs well.

Experiment 2: Video quality first vs. Bitrate first. We aim to evaluate QARC with different QoE parameters and the baseline algorithm which uses the policy based on high video bitrate. Specifically, we compare QARC to the baseline scheme in terms of queuing delay, the sending rate, and the video quality of the entire video session. The baseline is an offline optimal solution that uses greedy algorithm to reach the maximum bitrate for each time-slot during the session.

As shown in Figure 12 and Figure 13, compared with the baseline algorithm on broadband and 4G network environments, QARC outperforms the baseline scheme. In the broadband network environment, despite a shrinkage in average video quality of 4% - 9%, QARC decreases the sending rate of 46% to 60% and reduces the average queuing delay [5] from $0.5s$ to $0.04s$. It is noteworthy that if the footage of the video does not switch violently (Figure 12(b)), for instance, in video conference scenario, sending bitrate decreases from 51% to 62% while video quality reduces less than 5%. We can also find similar results in 4G network environments. Details can be seen in Figure 12.

[4]This experiment is worked on Azure with an instance in 20 CPUs and 140G RAM size.

[5]In this paper, queuing delay is regarded as self-inflicted delay, which is a lower bound on the 95% end-to-end delay that must be experienced between a sender and receiver, given observed network behavior. [34]

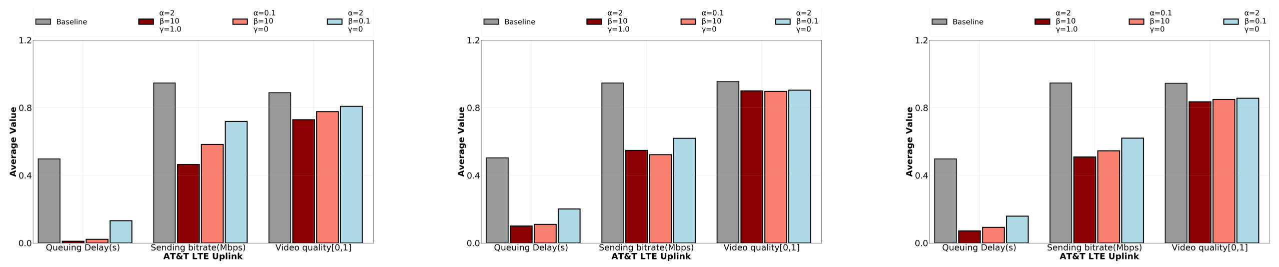


Fig. 12. Comparing QARC with different QoE and the baseline which is computed as an offline optimal value based on high video bitrate. We evaluate several QARC methods and a baseline on the **boardband network environments**. Like the process of Figure 11, after testing three video clips, results are shown as average queuing delay, average sending rates, and average video quality which are against the performance of the baseline value.

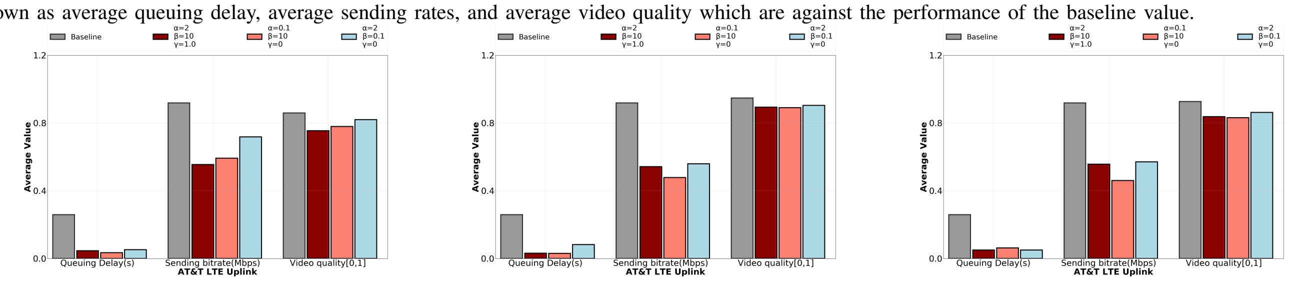


Fig. 13. Like the process of Figure 12, comparing QARC with different QoE and the baseline which is computed as an offline optimal value based on high video bitrate. We evaluate several QARC methods and a baseline on the **4G network environments**.

VI. Learning From QARC

In this section, we try to explore the insight of QARC by leveraging the attention heatmap of VQPN and VQRL respectively. The section is mainly composed of three parts, representing the learning process of each module in QARC. We finally propose an advanced QARC approach based on the insights of each module.

A. Learning From VQPN

Methodology: Unlike the previous work (e.g. [17]), VQPN extracts its attention features with jointly consider both spatial information and temporal information. Thus, the attention heatmap $\mathcal{A}_{i,k}$ for each frame f_i in VQPN's input is computed as

$$\mathcal{A}_{i,k} = \mathcal{F}^k(f_i) \cdot [\alpha_i w_k]^T. \tag{19}$$

Here, let $\mathcal{F}^k(f_i)$ represent the activation feature map of the last convolutional layer, and w_k reflects the weight corresponding to class k for output bitrates.

Visualization Setup: To better understand the reason why VQPN performs well, we apply our VQPN model to predict video quality on several video clips with different scenarios including a live cast scenario, a sports game video clip and a music video shot. Meanwhile, the attention feature map $\mathcal{F}(f_i)$ and the alignment vector α_i will be recorded during the test. Figure 14 summarizes the results.

Results and Analyze: As expected, the temporal information can be explained clearly through the results. We observe that the attention weight α perform quite different in various video scene types. In Figure 14(a), we find that in the form of the scene that is not frequently switched, VQPN stably refers to the spatial information of the past video frames, and vice versa (See Figure 14(h)). Due to the same video scene switching attribute, the remaining two video clips (Figure 14(g) , Figure 14(f)) show almost similar results.

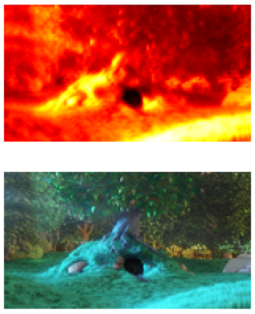
TABLE III
Comparing performance (RMSE) of Grayscale approach with VQPN.

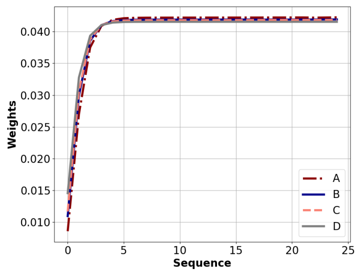
Video Clip Name	VQPN	Grayscale + GRU
Be yourself	**0.013**	0.026
Big Buck Bunny	**0.116**	0.116
Tsubasa wa Iranai	0.042	**0.040**
World Cup Live	**0.121**	0.124

The raw video frame and its attention heatmap are illustrated in Figure 14(a-d). However, we observe that there are some isomorphic features among the spatial information of video frames. No matter how complex the video frame is, the neural network still seems like to output the heatmap as a *grayscale* image rather than highlights the specific discriminative regions. Figure 15 shows the same conclusion clearly.

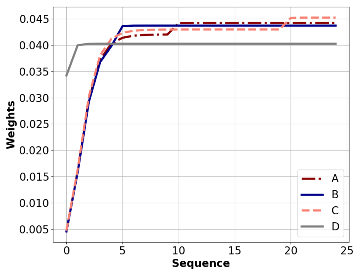
Rethinking VQPN's architecture: Our key idea is to identify the performance of the model using a group of grayscale video frames and a one-layer RNN. In detail, we use the average value of grayscale image to refer to the feature for each video frame, we then perform a one-layer GRU with a self-attentional model. The input and the output are similar to those of VQPN. In Table III, we report the results of this approach with different video clips and we also provide comparisons on RMSE between the grayscale approach and VQPN proposed in this paper. We observe that the results of two approaches are close, especially in the "Tsubasa wa Iranai" task, the grayscale approach performs better than VQPN with the improvement of 4.76%. Nevertheless, there still exists a wide distance in the static video scenes ("Be yourself").

(a) “Be yourself” [1].


(c) “Big Buck Bunny” [3].

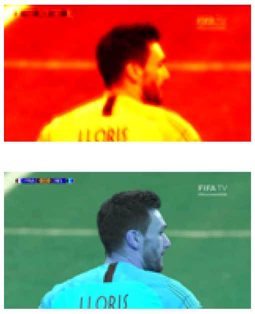


(e) The sample attention weight α_t of music video clip in static scenario (“Be yourself” [1])

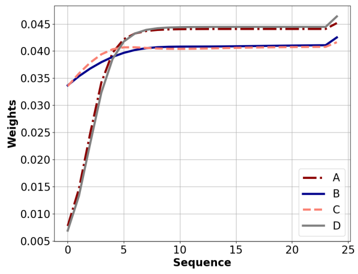


(g) The sample attention weight α_t of comic video clip (“Big Buck Bunny” [3]).

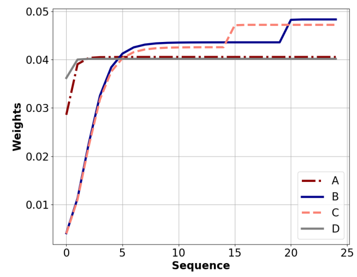
(b) “Tsubasa wa Iranai” [2].



(d) “World Cup Live” [4].

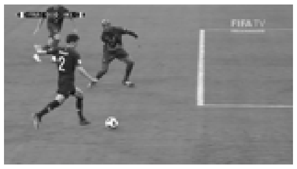


(f) The sample attention weight α_t of a music video clip within both static and dynamic scene (“Tsubasa wa Iranai” [2]).



(h) The sample attention weight α_t of a soccer game video clip (“World Cup Live” [4]).

Fig. 14. This group of figures shows our motivation: In the real-time live streaming scenario, high video bitrate is equaled to high video quality, however, in some circumstance, high video quality only requires a low bitrate.


(a) Raw Video Frame.

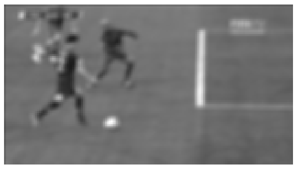

(b) Attention heatmap.

Fig. 15. The raw video frames and its attention heatmap are processed in grayscale image. The video frame is captured from “World Cup Live”.

Generalization: In this subsection, we first introduce the method of how to inspect the attention heatmap. We then discover temporal information insight through the attention heatmap and we further find that extracting the feature as the average value of a grayscale can replace the original CNN-based spatial information extraction model.

B. *Learning from VQRL*

Methodology: As is already described in IV-B, we use a network architecture similar to class activation maps(CAM) [17] to solve the bitrate selection task and to further understand its action. We, therefore, formulate the attention heatmap of VQRL as follows:

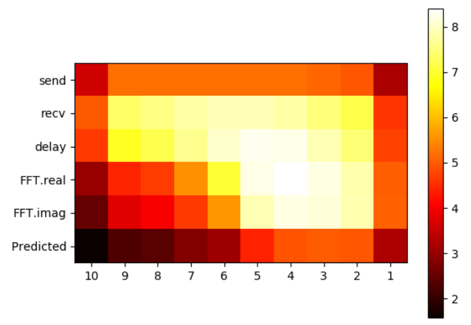
$$\mathcal{A}_c(S_t) = \sum_k w_k^c \mathcal{F}^k(S_t). \tag{20}$$

In which $\mathcal{A}_c(S_t)$ are defined as the attention heatmap for action c, $\mathcal{F}^k(S_t)$ represents the activation feature map of the last convolutional layer, w_k^c is the weight corresponding to action c for unit k and k means the filter number.

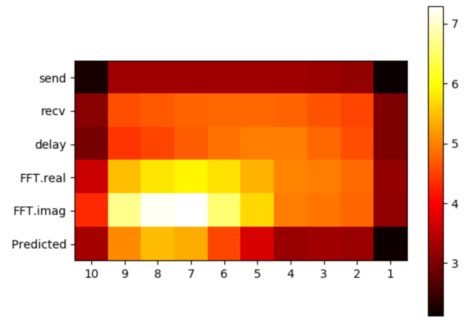
Experiments and Visualization Results: We visualize the attention heatmap of VQRL under different network conditions. Results are summarized in Figure 16. We notice that in stationary network conditions, VQRL focuses on almost all the network features including receive data and queuing delay gradient. By contrast, it neglects the video quality metric feature because of the feasible network conditions. However, while in the non-stationary network environments, VQRL pays more attention to the FFT features and the video quality metrics instead of the raw network status. This experiment also proves the importance of FFT characteristics in our work.

C. *Advanced QARC*

On the basis of the results of VQPN and VQRL, we move forward to investigate how to improve or “refine” the proposed QARC architecture. Inspired by the VQPN’s insight, we refine the VQRL’s network architecture by replacing the prediction video quality features to the average value of a grayscale image and disable VQPN module. In detail, the state of VQRL has been changed to $s_t = \{g, s, r, d, f_r^r, f_r^i\}$, in which g represents the average value of grayscale images in time-slot t (See Eq. 21, where $\mathcal{G} = [0.30, 0.59, 0.11]$), and the rest of them



(a) Attention heatmap of VQRL in stationary network environments.



(b) Attention heatmap of VQRL in non-stationary network environments.

Fig. 16. Attention heatmap of VQRL which are performed in both stationary and non-stationary network environments.

TABLE IV
COMPARISONS WITH DIFFERENT QARC APPROACHES ON THE SAME TEST SET. WE EVALUATE THE AVERAGE QOE AND THE OVERHEAD FOR EACH APPROACH RECEPTIVELY, AND THE BEST RESULT IS HIGHLIGHTED IN BLACK.

Scheme	Average QoE	Model Size (MB)
QARC (previous)	5.55 ± **0.34**	8.9
QARC	5.65 ± 0.74	6.5
Advanced QARC	**5.70** ± 0.92	**0.88**

are same as the state of previously proposed VQRL.

$$g = \frac{1}{W \times H \times N} \sum_{k=0}^{N} \sum_{x=0}^{W} \sum_{y=0}^{H} \mathcal{G}^T f_k(x, y) \tag{21}$$

We then implement the advanced QARC approach refer to the same action and reward as illustrated in Section V-B2. Next, we evaluate them on the same test dataset. Results are summarized in Table IV. By means of results, we find that the advanced QARC scheme obtains the close performance as the previous approaches by reducing about 90% of the model size overhead, thus we can prove that the conclusion learning from deep learning is effective and impressive.

VII. Discussion

A. Leverage 2D-Conv to take place of 1D-Conv in VQRL's representation

The key idea of improving the QARC's performance is to change 1D-Conv layers to 2D-Conv layers. In the previous scheme, temporal features can be extracted by 1D-Conv layer, however in the same time-slot, the informations of each factor are highly correlated with each other. Szegedy *et al.* [43] uses a $n \times 1$ convolution followed by a $1 \times n$ convolution equals to sliding a two layer network with the same receptive field as in a $n \times n$ convolution. Revisiting the above challenges, we perform several 2D-Conv layers to implement VQRL. Besides that, using 2D-Conv layers can also help us explain the neural network model.

B. Adding FFT features into VQRL's input

Research shows that training time of neural network can be reduced by pre-processing input data, such as pre-extracting features that be useful to human-beings [44]. We firstly assume receiving bitrate as a form of signal. Then, the Fast Fourier Transform (FFT) can be used to decompose signals into a complex-valued function of frequency, whose absolute value represents the amount of that frequency present in the original function, whose complex argument is the phase offset of the basic sinusoid in that frequency. [45]

Despite transforming time-series data to sequence data via FFT, an algorithm which computes the DFT of a sequence, is widely used for many schemes in various fields, such as fault diagnosis, engineering, it is still rarely used in describe the network conditions, especially, in live video streaming field. Moreover, the experiment of VQRL's visualization further prove the opinion. As a result, we add FFT features into state's input.

C. Influence of α,β and γ

Figure 12 and Figure 13 show the results of QARC with different initial QoE reward parameters. Unsurprisingly, initialize QoE reward with small latency coefficient α yield high-performance improvement over the one with a bigger α in wired network conditions, however, in 4G network environments, it performs a very different performance. In conclusion, there is no optimal pair can fit any network conditions.

D. Differ from the previous DASH works.

QARC is absolutely different from DASH works such as *Pensieve* and *D-DASH* [46].

- These papers perform well in the video-on-demand(VOD) scenario, which ignores the essential metric *latency*(re-buffering is only a part of latency). However, in the real-time video streaming scenario, latency is quite an important metric for evaluating QoE(Section V-A), and how to build an accurate network simulator(Section IV-C) is also one of the key contributions of our work. The reason why we use delay gradient method is described in Section IV-B.

- The motivation of *Pensieve* is also based on “bitrate first”, and it trains its model without considering video quality. *D-DASH* considers quality as an SSIM value, which is not sufficient to illustrate the insight of video, and they also ignore the trade-off between video quality and sending rate.

As a result, DASH works cannot perform well in our scenario. Meanwhile, live streaming with DASH can be concerned as the special case of QARC.

VIII. Related Work

In this section, we categorize the related work into three subgroups: real-time congestion control method, video quality metrics and deep reinforcement learning approaches.

A. Real-time rate control methods

Real-time rate control methods have been proposed and applied about two decades. These schemes are mainly classified into three types: loss-based bitrate approach, delay-based bitrate approach and model-based bitrate approach.

Loss-based: Loss-based approaches such as TFRC [5] and rate adaptation protocol (RAP) [6], have been widely used in TCP congestion control, and these methods increase bitrate till packet loss occurs, which means that the actions are always late, because when packet loss occurs, latency also increases. Thus, a bad experience will be given to the users because of the network jitter like sawtooth. Furthermore, using packet loss event as the control signal may cause its throughput to be unstable, especially in error-prone environments [47].

Delay-based: Delay-based approaches, try to adjust sending rate to control the transmission delay, can be divided into the end-to-end delay (RTT) approaches, such as TCP Vegas [7]; one-way delay approaches, such as LEDBAT (Over UDP) and TCP-LP [8], [48], and delay gradient approaches [9]. However, it is hard for the delay metrics to converge to the target value in some network conditions in which its upper limit has always suffered a wide range of changes, such as WIFI [47], [49].

Rate-based: These approaches controls the bitrate using the bandwidth estimation method which motivated by the adaptive filter algorithm. A large scale of adaptive filter algorithm has been used in bandwidth estimation, such as ARIMA, RLS, Kalman Filter, and so forth. The key limitation of these approach are shown as follows: 1. the result of bandwidth-based approach largely depends on the accuracy of bandwidth estimation method 2. the adaptive filter algorithm estimates the bandwidth by using fixed initial parameters which fits the specific network conditions, thus, it doesn’t have an ability to adapt to all network status.

Model-based: Model-based bitrate control method, such as Rebera [10] and GCC [9], they control sending bitrate based on previous network status observed including end-to-end latency, receiving rate which is measured by the receiver, and past sending bitrate, loss ratio measured by the sender side. In these schemes, After receiving the packet train, receiver periodically measures the queuing delay of each packet, then the receiver feeds the information of queuing delay and bytes received back to the sender every δ_t via another specific feedback channel. For the receiver side bitrate controllers [10], they control sending bitrate without any information from the sender side. After deciding the next period sending bitrate, the receiver side sends the value to the sender side via feedback message channel.

B. Video Quality Metrics

Video quality is a characteristic to measure the perceived video degradation while passing through a video transmission system. Up to now, the video quality metrics which are commonly used are shown as follows.

PSNR: A traditional signal quality metric [50], which is directly derived from mean square error (MSE) or its square root (RMSE). Due to the simplicity and low complexity of its calculation, PSNR continues to be the most popular evaluation of the video quality. However, the result cannot precisely reflect the visual quality seen by human eyes.

SSIM: An image quality metric, submitted in 2004 by Wang et al. [51]. Unlike previously proposed video quality evaluation criteria, SSIM uses the structural distortion measurement instead of mean square error. Due to the consideration of the whole picture, SSMI can give a properer evaluation of the video quality experienced by users. However, SSIM is not a professional tool for video quality assessment.

VMAF: Video Multi-method Assessment Fusion (VMAF) [18] is an objective full-reference video quality metric which is formulated explicitly by Netflix to estimate subjective video quality, in terms of a reference and distorted video sequence. Using machine learning techniques, VMAF provides a single output score in the range of $[0, 100]$ per video frame. In general, this metric is focused on describing the quality degradation due to compression and rescaling and it is closer to people’s real experience of video quality than previous schemes.

C. Deep Reinforcement Learning in the network approach:

In recent years, machine learning technologies have made breakthroughs in a large scale of fields. For the ability to construct models and making decisions directly with raw input data, they do not rely on any pre-defined rules. Reinforcement learning is generated to maximize the reward of each action taken by the agent in given states per steps. Recent years, several attempts have been made to optimize the network control algorithm using the reinforcement learning approach due to the difficulty of designing a fixed method to handle all network conditions.

Remy: Remy [52] decides with “a tabular method”, and it collects experience from the network simulator with network assumptions, however, like all TCP variants, when the real network deviates from Remy‘s input assumption, performance degrades.

Pensieve: Mao *et al.* [23] develop a system that uses deep reinforcement learning to select bitrate for future video chunks. Unlike most of the adaptive bit rate(ABR) algorithms, Pensieve does not need any predefined rules and assumptions to make decisions, and it can automatically adjust itself to the change of network conditions. By comparing with the existing ABR algorithms, Pensieve performs very well.

D-DASH: Gadaleta *et al.* [53] uses Deep-Q-learning method to perform a comprehensive evaluation based on state-of-the-art algorithms, including heuristics and learning-based, to evaluate performance metrics such as image quality and freeze/rebuffering events for video clips.

IX. Conclusion

We propose QARC, a deep reinforcement learning based rate control algorithm in the real-time live streaming scenario. Unlike previously proposed approaches, we try to obtain a higher video quality with possibly lower sending rate. Due to that heuristic algorithms cannot effectively handle the complicated scenarios caused by perplexing network conditions and various video content, we leverage deep reinforcement learning to select the future video bitrate, which can adjust itself automatically to the change of its inputs. To reduce the state space of the reinforcement learning model, we derive the neural network into two parts and train them respectively. After training on a board set of network data, We find that QARC outperforms existing rate control methods. We then consider to explain the QARC's insight by reconstructing its neural network. Based on the insights of QARC we observed, we significantly improve the performance of QARC.

Additional research may focus not only on the real-time live streaming scenario but also in 360° live video streaming, a tile-based transmission task.

Acknowledgement

We thank the anonymous reviewers for their valuable feedback. This work was done in close cooperation with Kuaishou Technology Co., Ltd.. The work is supported by the National Natural Science Foundation of China under Grant No. 61472204 and 61521002, Beijing Key Laboratory of Networked Multimedia No. Z161100005016051, and Key Research and Development Project under Grant No. 2018YFB1003703.

References

[1] R. RECORDS, "Genie chuo, be yourself," 2015. [Online]. Available: https://www.youtube.com/watch?v=GLGpsMvp7Jo

[2] J. T. O. Channel, "Jolin tsai, i'm not yours," 2015. [Online]. Available: https://www.youtube.com/watch?v=C7wRb9adQUc

[3] B. Foundation, "Big buck bunny, sunflower version," 2017. [Online]. Available: http://bbb3d.renderfarming.net/download.html

[4] FIFATV, "France v belgium - 2018 fifa world cup russia," 2018. [Online]. Available: https://www.youtube.com/watch?v=ntQsMSuEbyg

[5] M. Handley, S. Floyd, J. Padhye, and J. Widmer, "Tcp friendly rate control (tfrc): Protocol specification," Tech. Rep., 2002.

[6] R. Rejaie, M. Handley, and D. Estrin, "Rap: An end-to-end rate-based congestion control mechanism for realtime streams in the internet," in *INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 3, Mar 1999, pp. 1337–1345 vol.3.

[7] L. S. Brakmo and L. L. Peterson, "Tcp vegas: End to end congestion avoidance on a global internet," *IEEE Journal on selected Areas in communications*, vol. 13, no. 8, pp. 1465–1480, 1995.

[8] D. Rossi, C. Testa, S. Valenti, and L. Muscariello, "Ledbat: The new bittorrent congestion control protocol." in *ICCCN*, 2010, pp. 1–6.

[9] G. Carlucci, L. De Cicco, S. Holmer, and S. Mascolo, "Analysis and design of the google congestion control for web real-time communication (webrtc)," in *Proceedings of the 7th International Conference on Multimedia Systems*. ACM, 2016, p. 13.

[10] E. Kurdoglu, Y. Liu, Y. Wang, Y. Shi, C. Gu, and J. Lyu, "Real-time bandwidth prediction and rate adaptation for video calls over cellular networks," in *Proceedings of the 7th International Conference on Multimedia Systems*. ACM, 2016, p. 12.

[11] C. Z. L. S. Tianchi Huang, Rui-Xiao Zhang, "Qarc: Video quality aware rate control for real-time video streaming based on deep reinforcement learning," 2018. [Online]. Available: https://gitee.com/godka/startingpaper/raw/master/mm18.pdf

[12] E. M. Clarke, W. Klieber, M. Nováček, and P. Zuliani, *Model Checking and the State Explosion Problem*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 1–30.

[13] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International Conference on Machine Learning*, 2016, pp. 1928–1937.

[14] O. Biran and C. Cotton, "Explanation and justification in machine learning: A survey," in *IJCAI-17 Workshop on Explainable AI (XAI)*, 2017, p. 8.

[15] D. Gunning, "Explainable artificial intelligence (xai)," *Defense Advanced Research Projects Agency (DARPA), nd Web*, 2017.

[16] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio, "Show, attend and tell: Neural image caption generation with visual attention," in *International conference on machine learning*, 2015, pp. 2048–2057.

[17] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Learning deep features for discriminative localization," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2921–2929.

[18] R. Rassool, "Vmaf reproducibility: Validating a perceptual practical video quality metric," in *Broadband Multimedia Systems and Broadcasting (BMSB), 2017 IEEE International Symposium on*. IEEE, 2017, pp. 1–2.

[19] M. Inc., "x264 encoder," https://x264.org/, 2006.

[20] L. Merritt and R. Vanam, "x264: A high performance h. 264/avc encoder," *online] http://neuron2. net/library/avc/overview_x264_v8_5. pdf*, 2006.

[21] x265.org, "The x265 website," https://x265.org/, 2015.

[22] A. for Open Media, "AV one encoder," https:/aomedia.org/, 2018.

[23] H. Mao, R. Netravali, and M. Alizadeh, "Neural adaptive video streaming with pensieve," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 2017, pp. 197–210.

[24] N. Sato, T. Oshiba, K. Nogami, A. Sawabe, and K. Satoda, "Experimental comparison of machine learning-based available bandwidth estimation methods over operational lte networks," in *Computers and Communications (ISCC), 2017 IEEE Symposium on*. IEEE, 2017, pp. 339–346.

[25] S. Cappallo and C. G. M. Snoek, "Future-supervised retrieval of unseen queries for live video," 2017, pp. 28–36.

[26] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv: Neural and Evolutionary Computing*, 2014.

[27] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *CoRR*, vol. abs/1409.0473, 2014. [Online]. Available: http://arxiv.org/abs/1409.0473

[28] M. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," *CoRR*, vol. abs/1508.04025, 2015. [Online]. Available: http://arxiv.org/abs/1508.04025

[29] Z. Lin, M. Feng, C. N. d. Santos, M. Yu, B. Xiang, B. Zhou, and Y. Bengio, "A structured self-attentive sentence embedding," *arXiv preprint arXiv:1703.03130*, 2017.

[30] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998, vol. 1, no. 1.

[31] M. T. Hagan, H. B. Demuth, M. H. Beale *et al.*, *Neural network design*. Pws Pub. Boston, 1996, vol. 20.

[32] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[33] T. Huang, R. Zhang, C. Zhou, and L. Sun, "Delay-constrained rate control for real-time video streaming with bounded neural network," in *Proceedings of the 28th ACM SIGMM Workshop on Network and Operating Systems Support for Digital Audio and Video, NOSSDAV*

2018, Amsterdam, Netherlands, June 12-15, 2018, 2018, pp. 13–18. [Online]. Available: http://doi.acm.org/10.1145/3210445.3210446

[34] K. Winstein, A. Sivaraman, and H. Balakrishnan, "Stochastic forecasts achieve high throughput and low delay over cellular networks," pp. 459–471, 2013.

[35] R. Netravali, A. Sivaraman, S. Das, A. Goyal, K. Winstein, J. Mickens, and H. Balakrishnan, "Mahimahi: accurate record-and-replay for http," pp. 417–429, 2015.

[36] H. Wang, I. Katsavounidis, J. Zhou, J. Park, S. Lei, X. Zhou, M.-O. Pun, X. Jin, R. Wang, X. Wang, Y. Zhang, J. Huang, S. Kwong, and C.-C. J. Kuo, "Videoset: A large-scale compressed video quality dataset based on jnd measurement," vol. 46, 01 2017.

[37] M. F. B. Report, "Raw Data Measuring Broadband America 2016," https://www.fcc.gov/reports-research/reports/measuring-broadband-america/raw-data-measuring-broadband-america-2016/, 2016, [Online; accessed 19-July-2016].

[38] H. Riiser, P. Vigmostad, C. Griwodz, and P. Halvorsen, "Commute path bandwidth traces from 3g networks: analysis and applications," pp. 114–118, 2013.

[39] I. Telecommunications, "R-rec-bt.1788," https://www.itu.int/rec/R-REC-BT.1788-0-200701-I/en, 2007.

[40] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[41] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning." in *OSDI*, vol. 16, 2016, pp. 265–283.

[42] S. Ha, I. Rhee, and L. Xu, "Cubic: a new tcp-friendly high-speed tcp variant," *ACM SIGOPS operating systems review*, vol. 42, no. 5, pp. 64–74, 2008.

[43] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826.

[44] S. Li, G. Liu, X. Tang, J. Lu, and J. Hu, "An ensemble deep convolutional neural network model with improved ds evidence fusion for bearing fault diagnosis," *Sensors*, vol. 17, no. 8, p. 1729, 2017.

[45] M. Frigo, "A fast fourier transform compiler," *programming language design and implementation*, vol. 34, no. 5, pp. 169–180, 1999.

[46] M. Gadaleta, F. Chiariotti, M. Rossi, and A. Zanella, "D-dash: A deep q-learning framework for dash video streaming," *IEEE Transactions on Cognitive Communications and Networking*, vol. 3, no. 4, pp. 703–718, Dec 2017.

[47] Y. Geng, X. Zhang, T. Niu, C. Zhou, and Z. Guo, "Delay-constrained rate control for real-time video streaming over wireless networks," in *Visual Communications and Image Processing (VCIP), 2015*. IEEE, 2015, pp. 1–4.

[48] A. Kuzmanovic and E. W. Knightly, "Tcp-lp: low-priority service via end-point congestion control," *IEEE/ACM Transactions on Networking (TON)*, vol. 14, no. 4, pp. 739–752, 2006.

[49] K. Nihei, H. Yoshida, N. Kai, D. Kanetomo, and K. Satoda, "Qoe maximizing bitrate control for live video streaming on a mobile uplink," in *2017 14th International Conference on Telecommunications (ConTEL)*, June 2017, pp. 91–98.

[50] A. Hore and D. Ziou, "Image quality metrics: Psnr vs. ssim," pp. 2366–2369, 2010.

[51] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, April 2004.

[52] K. Winstein and H. Balakrishnan, "Tcp ex machina: computer-generated congestion control," *acm special interest group on data communication*, vol. 43, no. 4, pp. 123–134, 2013.

[53] M. Gadaleta, F. Chiariotti, M. Rossi, and A. Zanella, "D-dash: A deep q-learning framework for dash video streaming," *IEEE Transactions on Cognitive Communications and Networking*, vol. 3, no. 4, pp. 703–718, Dec 2017.

PLACE PHOTO HERE

Tianchi Huang received his M.E degree in the Department of Computer Science and Technology in Guizhou University in 2018.

His research interests lie in the area of rate control method, content delivery networks and multi-media.

PLACE PHOTO HERE

Rui-Xiao Zhang received his B.E degree in Electronic Engineering Department in Tsinghua University in 2017. Currently, he is pursuing his Ph.D candidate in Department of Computer Science and Technology, Tsinghua University, China.

His research interests lie in the area of content delivery networks, the optimization of multimedia streaming and cloud computing.

PLACE PHOTO HERE

Ming Ma is a PhD student in the Department of Computer Science and Technology at Tsinghua University, Beijing. Her research interests include content delivery networks and multi-media streaming optimizations. Ma has a B.E in computer science from Xidian University, Xian, China. Contact her at mm13@mails.tsinghua.edu.cn.

PLACE PHOTO HERE

Chao Zhou received the Ph.D. degree from the Institute of Computer Science & Technology, Peking University, Beijing, China, in 2014.He has been with Beijing Kuaishou Technology. From July 2014 to August 2015, he was a Senior Research Engineer with the Media Technology Lab, CRI, Huawei Technologies Co. Ltd., Beijing, China. His research interests include HTTP video streaming, joint source-channel coding, and multimedia communications and processing.

PLACE PHOTO HERE

Bing Yu received the B.E degree from Tsinghua University, Beijing, China. He has been with Beijing Kuaishou Technology.

PLACE PHOTO HERE

Lifeng Sun received the B.S and Ph.D degrees in system engineering from National University of Defense Technology, Changsha, Hunan, China, in 1995 and 2000, respectively. He joined Tsinghua University since 2001. He is currently a Professor with the Computer Science and Technology Department of Tsinghua University, Beijing.

Dr.Sun's research interests include the area of networked multimedia, video streaming, 3D/multiview video coding, multimedia cloud computing, and social media.