Buffer Awareness Neural Adaptive Video Streaming for Avoiding Extra Buffer Consumption

Tianchi Huang[1]*, Chao Zhou[2]*, Rui-Xiao Zhang[1], Chenglei Wu[1], Lifeng Sun[1,3,4]*
[1]Department of Computer Science and Technology, Tsinghua University [2]Beijing Kuaishou Technology Co., Ltd.
[3]BNRist, [4]Key Laboratory of Pervasive Computing (Tsinghua University), Ministry of Education, China
**Corresponding Authors.* {*htc19@mails.,sunlf@*}*tsinghua.edu.cn, zhouchao@kuaishou.com*

***Abstract*—Adaptive video streaming has already been a major scheme to transmit videos with high quality of experience (QoE). However, the improvement of network traffics and the high compression efficiency of videos enable clients to accumulate too much buffer, which might cause colossal data waste if users close the session early before the session ends. In this paper, we consider buffer-aware adaptive bitrate (ABR) mechanisms to overcome the above concerns. Formulating the buffer-aware rate adaptation problem as multi-objective optimization, we propose DeepBuffer, a deep reinforcement learning-based approach that jointly takes proper bitrate and controls the maximum buffer. To deal with the challenges of learning-based buffer-aware ABR composition, such as infinite possible plans, multiple bitrate levels, and complex action space, we design adequate preference-driven inputs, separate action outputs, and invent high sample-efficiency training methodologies. We train DeepBuffer with a broad set of real-world network traces and provide a comprehensive evaluation in terms of various network scenarios and different video types. Experimental results indicate that DeepBuffer rivals or outperforms recent heuristics and learning-based ABR schemes in terms of QoE while heavily reducing the average buffer consumption by up to 90%. Extensive real-world experiments further demonstrate the substantial superiority of DeepBuffer.**

I. Introduction

Video has proven itself to be even more significant than before due to periods of distancing and lockdowns for COVID-19 [1]. The Global Internet Phenomena Report 2022 [2] shows that from Jan. 2021 to June. 2021, the bandwidth traffic was dominated by streaming video, accounting for 53.72% of overall traffic, where YouTube [3], Netflix [4], and Facebook [5] video stand for the top three. Unsurprisingly, those three apps leverage adaptive video streaming for providing video services to the users, aiming to gain higher quality of experiences (QoE).

Client-based Adaptive bitrate (ABR) (or rate adaptation) schemes and techniques have been proposed to vary network conditions via picking the chunks with different bitrates [6]. Specifically, recent ABR approaches are motivated by predicting throughput [7], adjusting buffer occupancy [8], [9], or predefined model-free [10], [11] and model-based [12], [13], [14] ABR models. Each method, ideally, has its own advantages, such as high QoE performances [10], stable buffer control abilities [15], robust policies to avoid stall events [9], and varying diverse QoE requirements [16].

In this paper, we attempt to ask: *with the higher compression efficiency and sufficient bandwidth in the year 2022, what's the real challenge for today's ABR algorithms beyond gaining high performance?* With empirical analysis, we have observed that existing ABR algorithms immediately download each chunk once the previous chunk finishes downloading, which often occurs huge data waste if users stop watching videos unexpectedly (§II-B). Motivated by the success of conventional four-step ABR models [7], we consider jointly adjusting the maximum buffer size and the next chunks' bitrates to tackle the problem. Such maximum buffer policies allow ABR algorithms to wait for a while before downloading the next chunk, which can not only diminish the buffer overflow effect but also avoid unnecessary data wastage (§III-A).

Following the aforementioned mechanism, we model the buffer-aware rate adaptation as a multi-objective optimization problem. Then we convert it to the single-optimization using simple additive weighting (SAW) [17]. We propose DeepBuffer, a novel buffer-aware learning-based ABR algorithm. DeepBuffer trains a neural network (NN) model via state-of-the-art deep reinforcement learning (DRL) and synchronously controls the maximum buffer and next chunks' bitrate (§IV). To make DeepBuffer practical, we make several contributions, including NN's inputs, actions, as well as training methodologies. Firstly, beyond ABR's conventional metrics such as playback statics and video information, we further incorporate buffer preference, including current maximum buffer size and buffer weight, into the NN's input. Here the buffer weight ω is allowed to be dynamically adjusted w.r.t users' preference. For example, $\omega = 0$ means the user aims to achieve the highest QoE while paying little attention to the data wastage. Secondly, we design DeepBuffer's policy network with two outputs to decide bitrate action and max buffer action separately. Such settings can effectively reduce the action space for bootstrapping training. Thirdly, considering the diversity of video bitrate ladders, we propose a novel bitrate selection policy that can support the videos in the different number of bitrate levels with various encoded bitrate settings. In detail, we apply a gradient-based action mask behind the final output of the NN's bitrate selection layer, aiming to filter invalid actions that do not exist in the current bitrate ladder (§IV-C). Finally, to train DeepBuffer, we implement a novel sample-efficiency DRL method called Dual-Clip Phasic Policy Gradient (DCPPG). It combines several state-of-the-art on-policy DRL techniques, such as Dual-clip restriction algorithm [18] and auxiliary phasic policy method [19].

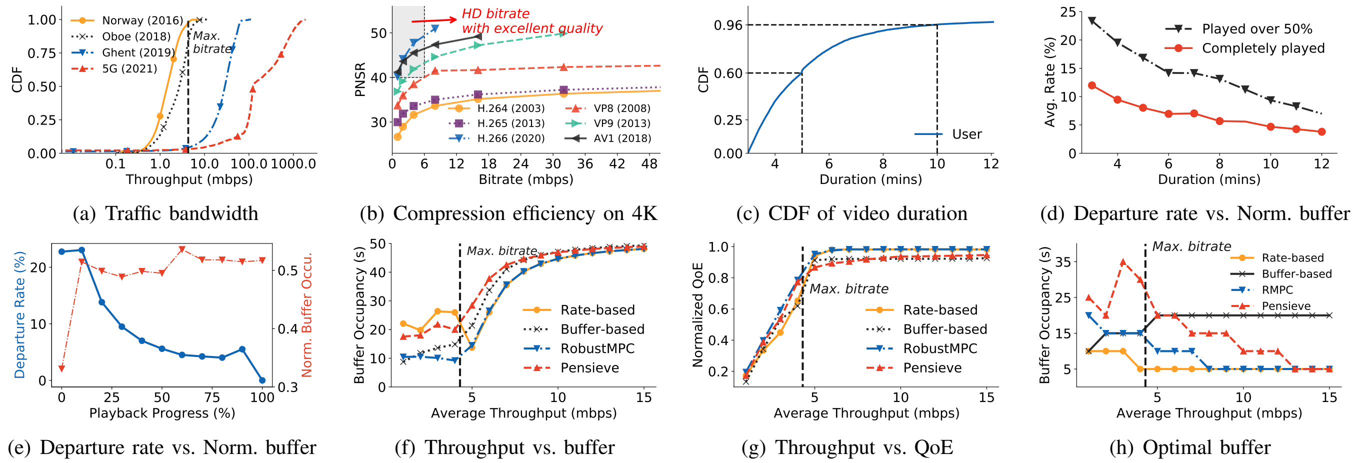


Fig. 1. *This group of pictures shows that the increased traffic bandwidth and everhigher compression efficiency result in the data wastage effect. The effect is becoming increasingly urgent in UGC-like services, since users often stop watching videos unexpectedly, then video data is lost in such sessions.*

We evaluate DeepBuffer with diverse video content and several real-world traces collected from various network conditions, categorized into slow-network, medium-network, and fast-network paths (§V). We first compare DeepBuffer in different buffer weights with state-the-art ABR algorithms involving heuristics, learning-based and wastage-based schemes. With trace-driven analysis, DeepBuffer shows its outstanding abilities in balancing QoE and buffer size, not only outperforming existing schemes by 1.8%-34.4% in terms of QoE over slow-network paths but also heavily reducing the buffer size up to 90% over fast-network paths. Next, DeepBuffer illustrates its high generalization abilities to varying multiple videos, where the videos have multiple types, pre-chunked with different bitrate ladders. Finally, we validate DeepBuffer over real-world network scenarios. Extensive results indicate the superiority of DeepBuffer against existing state-of-the-art approaches. In summary, our contributions are the following:

- We show how the data wastage problem affects today's ABR algorithms and how to solve it via buffer-aware adaptive video streaming. Then we address challenges to make the DRL-based scheme more practical (§II).
- We meticulously design the proper mechanism and train DeepBuffer with tailored NN architecture and methodologies (§IV).
- We comprehensively validate DeepBuffer with various videos and network settings, demonstrating multidimensional benefits of DeepBuffer in terms of QoE and buffer size (§V).

II. Background and Motivation

A. Related Work

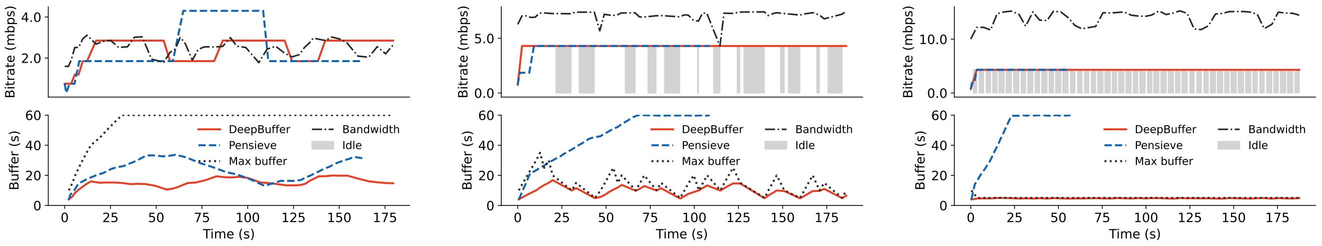
The history of ABR starts with heuristic methods. FESTIVE [20] and PANDA [7] make bitrate selection by estimating future throughput. BBA [8] and BOLA [9] are proposed to select bitrates w.r.t current buffer sizes. Then model-based approaches like MPC [14] leverage an offline ABR model for making decisions over a horizon. Altough several attempts [10], [12], [21] have been made to optimize the ABR algorithm based on various deep learning or RL methods, the above schemes seldom consider the data waste caused by unnecessary buffer accumulation.

The data-wastage effect has already been found for about one decade [3], [22], [23]. Especially, Plissonneau et al. [22] shows that recent ABR policies may lead to a large number of wasted bytes if the bandwidth is large enough. While in the past ten years, very little work has focused on solving such a dilemma. PSWA [24] is a wastage-based ABR algorithm for mobile video streaming. It controls the buffer solely with the offline trained configure map. Different from PSWA, DeepBuffer uses an NN-based policy to control both bitrates and maximum buffer with all considered metrics (§IV).

B. Motivation

We start by investigating how the buffer size influences traditional adaptive video streaming over today's network conditions. Figure 1(a) shows the explosive growth in network capacities in the past five years. As shown, almost 30× improvements in terms of the average bandwidth, ranging from 1.2 Mbps [25] to 300 Mbps [26]. However, Figure 1(b) shows encoding bitrate vs. PSNR (peak signal-to-noise ratio) plots for several generations of codec of two families – H.26x standards [27], [28], [29] and VPx groups [30], [31], [32]. We use a music video ([33], §V-A) with 4K resolution. Surprisingly, results indicates that the VP8 [30] and VP9 [31] can provide excellent quality at HD bitrates (6Mbps), as the latest advanced codecs such as AV1 [32] and H.266 [29] even performs well at SD bitrates (1.1Mbps) [34]. Thus, following the growth of compression efficiency, before ultra-video streaming like point-cloud and cloud-gaming becomes mainstream, conventional adaptive video streaming doesn't require rate adaptation logic over such increased traffic bandwidth – we can blindly pick the chunk with the highest bitrate throughout the entire session, and still, no stall events occur.

While beyond the sufficient bandwidth and better compression efficiency, we find that modern ABRs heavily suffer from data wastage problems. Specifically, ABRs often obey the “immediate download principle” that immediately downloads



(a) Slow-network path.

(b) Medium-network path.

(c) Fast-network path.

Fig. 2. *DeepBuffer is buffer-aware that not only picks excellent chunks for avoiding stalling but also focuses on reducing buffer occupancy in all considered network environments. Note the different behaviour of DeepBuffer in different types of network conditions, i.e., slow, medium and fast.*

the next chunk once the previous chunk has been downloaded [35]. Nevertheless, due to the overuse of the buffer, the client will not properly play all the video chunks downloaded if users leave prematurely, and eventually, resulting in the data wastage [20], [23]. We report user's measurement on Kuaishou [36], which covers over 100,000 unique videos with at least 3-minute duration. We find that almost 96% of videos have a duration of less than 10 minutes and 60% of videos perform less than 5 minutes (see in Figure 1(c)), while video completion rates are decreasing as the video duration increases (Figure 1(d)), from 12% for 3-minute videos to 5% for 12-minute videos. In other words, more than 92% of the users have never watched the end of the video. What's worse, more than 85% of the viewers have only watched less than half of the videos. Meanwhile, at the chunk level, Figure 1(e) shows that users will leave *at any chunk* and it's not related to the buffer occupancy.

Further, we conduct several experiments to verify the relationship between Quality of Experience (QoE, typically QoE$_{lin}$ [14]) and buffer occupancy of four popular ABRs over the network with an average bandwidth of 1-15Mbps. We use an HD video encoded by the maximum bitrate of 4.3 Mbps [37]. Figure 1(f) shows the average buffer size of each ABR algorithm starts to increase once the bandwidth is larger than the maximum encoded bitrate of the video. However, the increased buffer size doesn't actually improve QoE. Figure 1(g) shows that most ABR algorithms have achieved their optimal QoE value when the bandwidth reaches over 1.5× of the highest video bitrate. Hence, the client's buffer will be wasted if the bandwidth is sufficient for picking the highest bitrates, leading to unnecessary data costs.

To better understand how much buffer ABR algorithms waste in the video on demand (VOD) streaming, we measure the optimal buffer for each ABR scheme over different bandwidths and report the results curve in Figure 1(h), where the optimal buffer is the *offline* minimum buffer for keeping the optimal QoE score for each chunk. As expected, each algorithm shows a different optimal buffer under each bandwidth. The key reason is the value of the optimal buffer heavily depends on the strategy of ABR algorithms. For example, heuristics such as rate-based and buffer-based approaches either neglect or only consider buffer occupancy, which results in a relatively stable trend in the choice of the optimal buffer. By contrast, model-based and learning-based approaches such as RobustMPC and Pensieve use complex decision policies according to past throughput, buffer, and chunk size, which finally ramps down the optimal value slowly.

In summary, due to the rapid increases in traffic bandwidth, improvements in video compression efficiency, and diversity of user behaviors, it's critical to design a proper mechanism for avoiding data wastage, especially for videos with a duration of 3 to 10 minutes.

III. Methods

A. Joint buffer control and rate adaptation

To tackle the observation above, an intuitive idea of avoiding data wastage is to separate the rate adaptation scheme and the download scheduling scheme, just like PANDA [7]. Nevertheless, in the previous section (§II-B) we have shown that each ABR algorithm has its own optimal scheduling policy. In this work, we propose the concept of buffer-aware adaptive video streaming. Different from recent buffer-based approaches [8], buffer-aware ABR joints scheduling and rate adaptation scheme in *one step*. We schedule the next chunk's download time by controlling the maximum buffer size.

Figure 2 demonstrates the design principle of buffer-aware adaptive video streaming, which is absolutely different from previous work. In the slow-network path (Figure 2(a)), we do not have to adjust the maximum buffer, as the ABR policy can naturally preserve its buffer size while providing high QoE. In the medium-network path (Figure 2(b)), we partially control the maximum buffer and keep the current buffer within a proper range to avoid unnecessary data wastage during the session. Such scenario enables the algorithm to make *quantizing* and *scheduling* decisions just like PANDA [7]. Alternatively, since the bandwidth is quite sufficient in the fast-network path (Figure 2(c)), the maximum buffer controller dominates the process that entirely sets the buffer as the minimum buffer size. Most of the time, the download module is worked in idle states, receiving intermittent bandwidth information for estimation. To that end, both ABR policy and buffer control algorithm is non-trivial for buffer-aware ABR algorithm.

B. Buffer-aware rate adaptation model

We formally model the buffer-aware ABR. In the typical ABR video streaming, the videos are pre-chunked into a series of chunks, each of which is segmented as the same video time of L seconds. Assuming that there are M bitrate levels for a video with the bitrates of $R = \{R_1, R_2, \ldots, R_M\}$. Let B_t be the buffer occupancy at the start of downloading chunk t, R_t represent the selected video bitrate, C_t as average throughput measured, $d_t(\cdot)$ is the video chunk size for bitrate R_t.

Now, considering the maximum buffer size B_t^{max} as another policy that can be adjusted for chunk t, we extend the traditional process as *buffer-aware adaptive video streaming*. The buffer occupancy of the next chunk B_{t+1} can be concluded as Eq. 1. When the current buffer size has reached or "overflowed" B_t^{max}, the player will wait for the buffer to drain to a certain level which the next chunk $t + 1$ could be downloaded.

$$B_{t+1} = min(\left(B_t - \frac{d_t(R_t)}{C_t}\right)_+ + L, B_t^{max}). \tag{1}$$

By using this mechanism, the video player can "postpone" for a while to download chunks and, in turn, actively pick them at any time in the future. Such operations enable the players to maintain the current buffer level to avoid unnecessary playback buffer wastage. Further, we formulate the buffer-aware ABR problem as a multi-objective optimization problem, i.e., the combination of QoE maximization and buffer minimization problem, listed in Eq. 2, where δu_t represents the additional waiting time caused by Round-Trip-Time (RTT), render time, and especially, the maximum buffer threshold.

$$\max_{R_1,\ldots,R_N,T_s} \sum_t QoE_t^N, \min_{B_t^{max},T_s} \sum_t B_t \tag{2}$$

$$\text{s. t.} \quad u_{t+1} = u_t + \frac{d_t(R_t)}{C_t} + \delta u_t, \tag{3}$$

$$C_t = \frac{1}{u_{t+1} - u_t - \delta u_t} \int_{u_t}^{u_{t+1}-\delta u_t} C_n dn, \tag{4}$$

$$B_{t+1} = min\left[\left(B_t - \frac{R_t L}{C_t}\right)_+ + L - \delta u_t, B_t^{max}\right], \tag{5}$$

$$R_t \in \{R_1, R_2, \ldots, R_M\}. \tag{6}$$

Here we consider the buffer-aware ABR composition as the multi-objective optimization problem. In common, multi-objective optimization problems have no single solutions but a set of so-called Pareto-optimal solutions, which means none of the objective functions can be improved without degrading some of the other objective values. The set of Pareto-optimal solutions represents the tradeoffs according to all objective functions. Solving such multi-objective optimization means finding *all the possible solutions*.

C. Challenges for DRL-based approaches

We straightforwardly convert multi-objective optimization to single-optimization via the traditional method, namely simple additive weighting (SAW) [17]. The surrogate objective function is listed in Eq. 7, in which ω means how much the function is influenced by the buffer occupancy. Larger ω indicates that the ABR policy pays more attention to lower the buffer rather than preserving QoE performance, vice versa.

$$\max_{R_1,\ldots,R_N,B_t^{max}} \sum_t QoE_t^N - \omega B_t \tag{7}$$

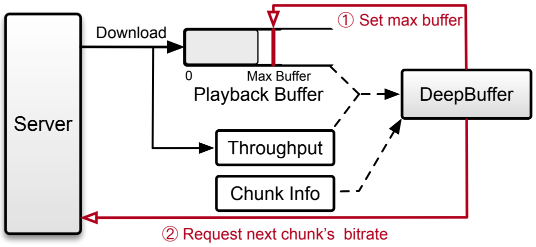


Fig. 3. *A high-level perspective of DeepBuffer. Different conventional ABR algorithms, buffer-aware ABRs both select proper bitrate and manage buffer.*

Taking the objective function as a *reward signal*, we can leverage the state-of-the-art deep reinforcement learning (DRL) method to generate ABR algorithms since the problem naturally falls into the scope of DRL. However, directly using the DRL method to solve the problem is impractical. We have to face several key challenges:

▷ ***"Infinite" possible objectives.*** Single-objective optimization only provides a single execution plan instead of all possible Pareto-optimal plans that exhibit the tradeoffs between the different plans. While in our buffer-aware ABR setting, there are infinite possible sets of plans, which become intractable. Thus, how to efficiently obtain all the solutions without exploring the whole objective space?

▷ ***Varying multiple videos.*** The buffer control strategy is highly influenced by the ratio between maximum bitrate and current throughput measured. Unfortunately, recent learning-based ABR schemes only support one bitrate ladder setting [12], [11], or vary all combinations of the ladders that only cover the entire DASH video list [10]. Hence, how to help learning-based ABRs tame the videos in such "real" yet multiple bitrate levels?

▷ ***Learning policies in complex action spaces.*** Besides, the action spaces contain two sub-actions, i.e., bitrate action and maximum buffer action. They perform independently. So how to let the learned ABR take the two actions effectively?

Putting them together, we have to i) implement sophisticated NN architecture for fulfilling variable feature inputs, ii) design adequate training methodology, and iii) propose a novel DRL algorithm that can provide greater sample efficiency in comparison to existing algorithms.

IV. DeepBuffer Design

To face the challenges above, we present DeepBuffer, a novel neural buffer-aware ABR algorithm. The big picture of DeepBuffer is demonstrated in Figure 3. Upon receiving the state representation, DeepBuffer adopts a NN to pick the proper bitrate and set the maximum buffer for the next chunk. In this section, we describe DeepBuffer's NN inputs, outputs, architecture, and training methodology.

A. NN overview

State. As shown in Figure 4, for each chunk t, the state input s_t is defined as: $s_t = \{N_t, V_t, \omega\}$, where N_t means video playback statics, V_t is the video content metric, and ω controls buffer preference metrics.

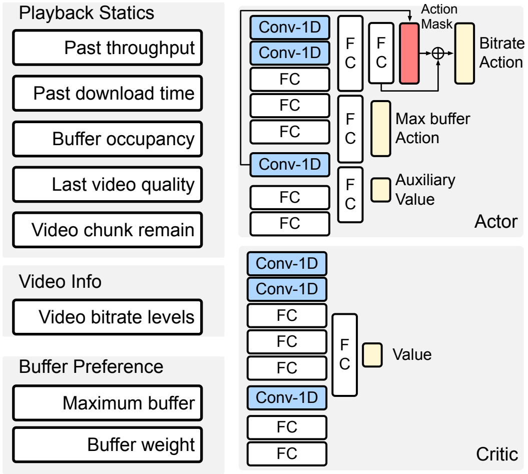


Fig. 4. *DeepBuffer's NN architecture. There are two action spaces in the actor network. An auxiliary value is given for accelerating the training process.*

▷ ***Video playback statics*** N_t. We take four critical metrics for describing the video playback status. The metric includes past bitrate selected q_t, current buffer occupancy b_t. Meanwhile, it also contains two sequences, i.e., past k chunks' throughput measured $C_t = \{c_{t-k+1}, \dots, c_t\}$ and download time $m_t = \{m_{t-k+1}, \dots, m_t\}$. All the metrics have been normalized within a proper range. We set k=8 [10].

▷ ***Video information*** V_t. DeepBuffer directly takes the *average bitrate* of each bitrate level as the video information. Different from previous work, we collect numerous "real videos" as the video set but instantly generate the "fresh video" with the *video generator* during training. The videos are chunked as four seconds [10], [38], [12]. Details of the video generator please check §V-A.

▷ ***Buffer metric*** ω. In practice, users and content providers may have different requirements for balancing QoE and buffer. For instance, some providers prefer high QoE services without considering the buffer wastage (i.e., $\omega = 0$), while others prefer fixing the maximum buffer size into a lower value to limit the overall bandwidth cost (i.e., $\omega = 1$). To express such application requirements, we take the current maximum buffer occupancy B_t^{max} and buffer weight ω into the state.

Actions. With the state s_t, the agent synchronously takes two actions, i.e., a_t and b_t, in which a_t is the next chunk's selected video bitrate and b_t reflects the next chunk's "relative" maximum buffer size (seconds). The maximum buffer size is updated as Eq. 8. We discuss DeepBuffer with different types of action spaces in §V-E.

$$B_t^{max} = B_{t-1}^{max} + b_t, b \in \{-10, -5, 0, 5, 10\}(seconds). \quad (8)$$

Reward. We set Eq. 7 as the DeepBuffer's reward function. Note the function is dynamically parameterized by the buffer weight during training. Such settings enable agents to better understand the correlation between the strategy and the requirement.

Architecture. The DeepBuffer's NN is composed of an actor network θ and a critic network θ_v. The actor network outputs the probabilities of bitrate action π_θ^a and maximum buffer action π_θ^b. The critic network outputs the value of the current state V_{θ_v}.

▷ **Actor network.** The DeepBuffer's actor network includes bitrate action, max buffer action, and the auxiliary value. All three actions and value uses a shared network to extract features from the given state. In detail, the actor's shared network adopts three 1D-convolution (Conv-1D) layers with feature number=128, kernel size=1 to extract the features from throughput, download time, and bitrate levels. The rest of the metrics are passed to five fully-connected (FC) layers with the same shape of 128 neurons. The resulted features are concentrated into a shared vector. For outputting the bitrate action, we use another FC layer with 128 neurons to down-sample the result of the shared network. The result of the layer then passes to an FC layer with the neuron number of $|A|$, which is the maximum dimension of the bitrate action. No active functions are applied. Here we treat the output as m. Due to the variable video bitrates and video count, we then apply an action mask to filter invalid actions. The mask can be implicitly estimated from video content features $V_t = \{v_t^0, \dots, v_t^i, v_t^{|A|}\}$ because we have already set the invalid indices to -1 w.r.t the bitrate level. Assuming $\mathbb{I}(\cdot)$ as a binary indicator, a_t can be sampled from the probability:

$$a_i \sim \frac{\mathbb{I}(v_t^i > 0)e^{m_i}}{\sum_{j \in |A|} \mathbb{I}(v_t^j > 0)e^{m_j}}. \quad (9)$$

Note that the standard backpropagation of the gradient in the NN still holds [39]. For representing the maximum buffer action, we take an $|B|$-dim vector with Softmax function after another feature down-sampling layer with 128 neurons, in which $|B|$ is the count.

Moreover, we output a single scalar named auxiliary value. The auxiliary value is used purely to train representations for the policy. It will be optimized by the auxiliary loss during the auxiliary phase.

▷ **Critic network.** We implement a critic network to output a value, which learns an estimate of the accumulate reward (i.e., total value) and helps improve the total performance of the actor network.

B. Policy optimization with DCPPG

We have to construct a *more* sample-efficient DRL algorithm to tame the complexity of the buffer-aware ABR task since the algorithm controls two actions with a variable action space and a fixed action space. Inspired by the recent success of auxiliary learning [19], we propose a novel DRL algorithm, namely Dual-Clip Phasic Policy Gradient (DCPPG). The DCPPG's training process is mainly composed of the policy phase and the auxiliary phase.

During the policy phase, we separately update the actor network and the critic network. In detail, the loss function of the DeepBuffer's actor network is trained by Dual-Clip Proximal Policy Optimization (Dual-PPO) [18], computed as $\mathcal{L}^{Policy}$ (Eq. 10):

$$\mathcal{L}^{Policy} = \hat{\mathbb{E}}_t[\mathbb{I}(\hat{A}_t < 0)\max(\mathcal{L}^{PPO}, c\hat{A}_t) + \mathbb{I}(\hat{A}_t \geq 0)\mathcal{L}^{PPO}] \quad (10)$$

Here $\mathbb{I}(\cdot)$ is a binary indicator function, $\mathcal{L}^{PPO}$ (Eq. 11) can be viewed as the surrogate loss function of the vanilla PPO,

$\rho(\pi_\theta)$ (Eq. 12) reflects the joint probability ratio of π_θ^a and π_θ^b, $\hat{A}_t$ (Eq. 13) is the advantage function that is learned by bootstrapping from the current estimate of the value function, and γ=0.99 represents the discounted factor. Briefly, the Dual-clip PPO algorithm adopts a double-clip method to restrict the step size of the policy iteration and update the NN by minimizing the *clipped surrogate objective*. ϵ and c are hyper-parameters that control how to clip the gradient. By default, we set $\epsilon = 0.2$, $c = 3$ [18].

$$\mathcal{L}^{PPO} = \min\left[\rho(\pi_\theta)\hat{A}_t, \text{clip}\big(\rho(\pi_\theta), 1 \pm \epsilon\big)\hat{A}_t\right] \qquad (11)$$

$$\rho(\pi_\theta) = \frac{\pi_\theta^a(a_t|s_t)}{\pi_{\theta_{\text{old}}}^a(a_t|s_t)} \cdot \frac{\pi_\theta^b(b_t|s_t)}{\pi_{\theta_{\text{old}}}^b(b_t|s_t)} \qquad (12)$$

$$\hat{A}_t = r_t + \gamma V_{\theta_v}(s_{t+1}) - V_{\theta_v}(s_t) \qquad (13)$$

Moreover, the parameters of the DeepBuffer's critic network θ_v are updated by minimizing the error of $\hat{A}_t$.

$$\nabla\mathcal{L}^{\pi} = -\nabla_\theta\left[\mathcal{L}^{PPO}(\pi_\theta^a, \pi_\theta^b, \hat{A}_t) + \lambda H_\theta(s_t)\right] + \nabla_{\theta_v}\left[A_t\right]^2. \quad (14)$$

We summarize the loss function $\mathcal{L}^{\pi}$ in Eq. 14. In addition, we add the entropy of all the policies $H_\theta(s_t)$ into the loss function to encourage exploration feedback, where λ is the entropy weight. Considering that on-policy RL is sensitive to the entropy weight [38], we adjust the entropy weight λ to minimize the gap between the current entropy and the target entropy H_{target}. Here, we set $H_{target} = 0.1$ [40].

During the auxiliary phase, we further optimize the actor network according to the joint objective function $\mathcal{L}^{joint}$ which includes behavioral cloning loss and an arbitrary auxiliary value loss:

$$\begin{aligned}\mathcal{L}^{joint} =& \hat{\mathbb{E}}_t\left[\mathbf{KL}\left(\pi_{\theta_{old}}^a(s_t), \pi_\theta^a(s_t)\right) + \mathbf{KL}\left(\pi_{\theta_{old}}^b(s_t), \pi_\theta^b(s_t)\right)\right] \\ &+ \hat{\mathbb{E}}_t\left[\frac{1}{2}\left[V_{targ}(s_t) - V_\theta(s_t)\right]^2\right].\end{aligned} \qquad (15)$$

Here $\mathbf{KL}(\cdot)$ is the behavioral cloning loss, representing the KL-divergence between the original policy (i.e., $\pi_{\theta_{old}}^a$, $\pi_{\theta_{old}}^b$) and the updated policy (i.e., π_θ^a, π_θ^b). Please note that the original policy here is the policy after the ending of the previous policy phase and just right before the beginning of auxiliary phase. The rest part of Eq. 15 is an auxiliary value function that minimizes the gap between target value $V_{targ}(s_t)$ and the auxiliary value $V_\theta(s_t)$, in which the target value is estimated by the combination of reward for the current state s_t and the value of the critic network for the next state s_{t+1}: $V_{targ}(s_t) = r_t + \gamma V_{\theta_v}(s_{t+1})$.

C. Training methodology

Alg. 1 presents the main phases for training DeepBuffer.

Phase 1: randomizing environments. The first phase proposes a randomized environment generator that fully considers the diversity requirements in terms of video bitrates, network information, buffer weights, etc. Specifically, we first randomly initialize the "fresh videos" with the video generator, as each video contains 2-6 bitrate levels. Next, we uniformly pick a trace from the network dataset. Finally, we randomize the weight ω to demonstrate the buffer requirement.

Algorithm 1 *DeepBuffer Training Process*

```
import random
network_pool = load_trace()
while not converge:
    #Phase 1: randomize environments
    #video count
    count = random.choice([2, 3, 4, 5, 6])
    #randomize bitrate ladders: 100-7000kbps
    video = sorted(random.uniform(100, 7000, count))
    size = video_generator(video) (§V-A)
    #randomly pick a network trace
    n_info = random.choice(network_pool)
    #randomize buffer weight
    $\omega$ = random.uniform(0, 1)
    #Phase 2: rollout policy
    #array: state, bitrate, buffer, reward
    $\mathbb{S}$, $\mathbb{A}$, $\mathbb{B}$, $\mathbb{R}$ = rollout(video, size, $\omega$, n_info)
    #Phase 3: training with DCPPG
    for _ in range($N_{policy}$):
        Optimize $\theta$, $\theta_v$ according to $\mathcal{L}^{\pi}$(Eq.14)
    $\pi^a_{old}$, $\pi^b_{old}$ = predict($\mathbb{S}$)
    for _ in range($N_{aux}$):
        Train $\theta$, $\theta_v$ using $\mathcal{L}^{joint}$(Eq.15), $\pi^a_{old}$, and $\pi^b_{old}$.
    $\lambda \mathrel{+}= \alpha(H_{target} - H_\theta(s_t))$ # Update entropy weight
```

Phase 2: rollout policy. In this phase, we encapsulate the buffer-aware ABR-process into a gym-like [41] environment, which allows the agent to learn the policy effectively.

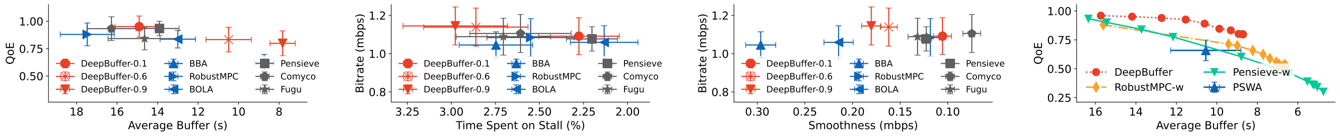
Phase 3: training with DCPPG. The actor network and the critic network are repeatedly and iteratively optimized by different objective functions in the policy phase and auxiliary phase. Here note that the old policies of bitrate π_{old}^a and buffer π_{old}^b should be estimated again just before the auxiliary phase starts. Now we briefly introduce the role of each hyperparameter. N_{policy} is the number of policy updates performed in each policy phase. N_{aux} controls the sample reuse during the auxiliary phase. We set N_{policy}=5, N_{aux}=6 with consistent of the original PPG paper [19].

V. Evaluation

A. Methodology

Experimental Setup. We employ trace-driven simulation with virtual player [42] and real-world evaluation (§V-F). We modify the player to enable maximum buffer adjustment. Each experiment runs for all segments in the video emulated over network traces.

Video generator. We propose a video generator that enables diverse videos with different encoding bitrates during training. Specifically, we select 86 videos from YouTube, which involves movies, sports, games, news, and MVs, and encode the video by H.264 codec according to the nine fixed bitrates and segment it into 4-second chunks. During training, we randomly initialize a bitrate ladder with the bitrate range from 100 to 7000 kbps in 2-6 levels. Then the video size can be estimated by the piece-wise linear-regression method – It's simple yet effective, as the proposed generator performs at least 15,000× acceleration with an accuracy of 98.83%.



(a) Buffer vs. QoE (b) Stall vs. Bitrate (c) Smoothness vs. Bitrate (d) DeepBuffer with different ω

Fig. 5. *Comparing DeepBuffer with recent ABR algorithms on the QoE_{lin} over the HSDPA dataset. Error bars show 95% confidence intervals.*

Video Test Sets. We adopt three video sets with different types of bitrate ladders for testing. i) *EnvivioDash3* [37]: the DASH.js [43] reference video which is encoded by six bitrates in the range of $\{0.3, 0.75, 1.2, 1.85, 2.85, 4.3\}$ Mbps. ii) *Tears of Steel* [44]: a short science fiction film encoded as $\{0.35, 0.6, 1, 2, 3\}$Mbps. iii) *WA DA DA* [33]: a K-pop music video (MV) proposed by Kep1er. The video is encoded as $\{0.4, 1, 3, 6\}$Mbps. All videos are encoded by the H.264 codec [27].

Network Trace Datasets. We use Puffer public dataset [13], which involves over 50,000 network traces, for training. Meanwhile, we organize recent public datasets into three categories for testing: i) slow-network paths ($\leq$6Mbps), including HSDPA [25] and FCC [45]; ii) medium-network paths ($\leq$100Mbps), containing Oboe [38] and FCC-18 [46]; iii) fast-network paths (>100Mbps), 5G [26].

QoE Metrics. In this paper, we employ two QoE metrics. The first is QoE$_{lin}$ [14], [38], [10], [13], the vanilla linear mapping function:

$$QoE_{lin} = \sum_{n=1}^{N} q(R_n) - \max q(R) \sum_{n=1}^{N} T_n - \sum_{n=1}^{N-1} |q(R_{n+1}) - q(R_n)| \,, \tag{16}$$

where N is the total number of chunks, R_n means the chunk n's video bitrate, T_n is the rebuffering time, $\max q(R)$ means the maximum bitrate of the bitrate ladder, $q(R_n)$ is a function that maps the bitrate R_n to the quality perceived by the user. Here we set $q(R_n) = R_n$ since it effectively helps us *analyze several underlying QoS metrics*, such as bitrate, stall, and smoothness. Ideally, we can use any form of perceptual measurement, such as SSIM [13] and VMAF [12], [21].

The second is QoE$_{itu}$, which is calculated by ITU-T Rec P.1203 [47]. We select QoE$_{itu}$ since it's a parametric bitstream-based quality assessment standard.

ABR Baselines. In this work, we select several representational ABR algorithms from various type of fundamental principles. For recent heuristics, we select the following ABR algorithms, marked as **blue**: i) ***Buffer-based Approach*** **(BBA)** **[8]:** a vanilla buffer-based approach; ii) ***BOLA*** **[9]:** the standard ABR scheme which solves the ABR problem by the Lyapunov function; iii) ***RobustMPC*** **(RMPC)** **[14]:** a model-based method by predicting key environment variables over a moving look-ahead horizon.

Learning-based ABR baselines includes (marked as gray): iv) ***Pensieve*** **[10]:** the vanilla DRL-based ABR scheme. We use the pre-trained model; v) ***Comyco*** **[12]:** a quality-aware imitation learning-based ABR scheme. We retrain it with QoE$_{lin}$; vi) ***Fugu*** **[13]:** a hybrid ABR algorithm that adopts deep neural network (DNN) to predict the download time for the next chunk and uses vanilla MPC to make decisions.

In addition, we also consider prior wastage-based ABRs as the baselines, including vii) ***Pensieve-ω*****:** a hybrid scheme that takes Pensieve as the basic ABR algorithm and uses a learned policy to control the maximum buffer. The policy is trained via maximizing Eq. 7. viii) ***RobustMPC-ω*****:** a heuristic that considers *all the possible bitrate-buffer action pairs* and maximizes QoE over a horizon of future five chunks like MPC. This scheme can be regarded as the upper bound of heuristics to deal with the buffer-aware adaptive video streaming problem. ix) ***PSWA*** **[24]:** a closest scheme compared with DeepBuffer. PSWA is the wastage-based ABR scheme that adjusts the buffer via a configured map, in which the map is pre-trained based on the epsilon-constraint method [48] and only takes past throughput as the input.

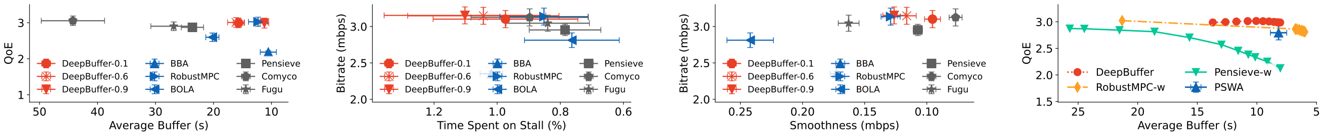
Implementation. DeepBuffer's training tools are built with TensorFlow 2.8.1 [49]. We set $|A|$=6 (i.e. max. six bitrate levels), $|B|$=5, learning rate $\alpha = 10^{-4}$, and train the model with QoE$_{lin}$. Note, §V-D shows QoE$_{itu}$ results.

B. DeepBuffer vs. existing ABR algorithms

In this part, we leverage trace-driven simulation to compare the performance of DeepBuffer against several existing ABR algorithms over various kinds of network types, including slow-network (HSDPA [25]), medium-network (FCC-18 [46]), and fast-network (5G [26]) paths. During the experiment, we only utilize the same trained model and set $\omega = 0.1, 0.6, 0.9$, i.e., DeepBuffer-ω. Results are tested over the Envivio video set and summarized as QoE$_{lin}$ (§V-A). We discuss DeepBuffer with different ω in §V-C.

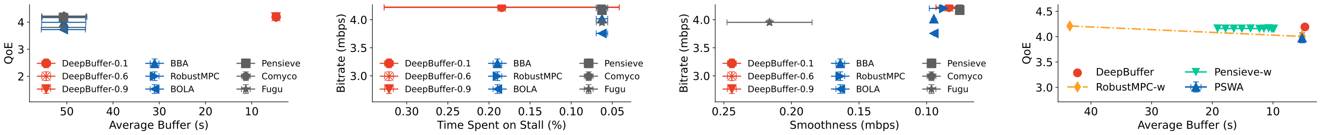
Slow-network paths. In Figure 5(a), DeepBuffer-0.1 achieves the best scheme on the slow-path network condition, with the improvements on average QoE of 1.8% (Comyco) - 34.4% (BBA) compared with existing ABRs. Meanwhile, we find that DeepBuffer-0.6 and DeepBuffer-0.9 not only gain acceptable overall performance but also reduce the average buffer size. As shown, DeepBuffer-0.6 rivals BOLA and Fugu on average QoE but heavily decreases 23.9% on BOLA and 39.6% on Fugu, respectively. Meanwhile, DeepBuffer-0.9 improves 21.8% on QoE compared with BBA, and it reduces 11.6% on buffer size.

Moreover, we report the detailed metrics on slow-network paths in Figure 5(b) and Figure 5(c). The results contain various critical metrics, such as average bitrate, stalling ratio, and average bitrate change (i.e., smoothness). Note the right top region of the figures is the desired operation region for any scheme. As shown, we claim that DeepBuffer's superior performance is due to its better understanding of the shape of *the Pareto frontier*, since DeepBuffer with various ω always performs in a Pareto optimum state when no bitrate or stall changes can make one individual better off without making at least one other individual worse off. Similarly, DeepBuffer



(a) Buffer vs. QoE (b) Stall vs. Bitrate (c) Smoothness vs. Bitrate (d) DeepBuffer with different ω

Fig. 6. *Comparing DeepBuffer with recent ABR algorithms on the* QoE_{lin}*. Results are collected on the FCC-18 dataset.*



(a) Buffer vs. QoE (b) Stall vs. Bitrate (c) Smoothness vs. Bitrate (d) DeepBuffer with different ω

Fig. 7. *Comparing DeepBuffer with recent ABR algorithms on the* QoE_{lin}*. Results are collected on the Lumos-5G dataset.*

also maintains its competitiveness in terms of the smoothness metric, which stands for the Top-2 scheme when ω=0.1.

Medium-network paths. We analyze the behavior of DeepBuffer and baselines over the FCC-18 network dataset. The dataset contains various network conditions, which can be viewed as nowadays' network. From Figure 6(a), we observe that existing learning-based approaches suffer from data wastage, which often leverages an additional 1.5× (Pensieve), 1.7× (Fugu), and 2.8× (Comyco) on buffer size compared with DeepBuffer for achieving similar results on QoE. By contrast, comparing DeepBuffer with recent heuristics such as BBA and BOLA, we can see that DeepBuffer-0.1 improves the average QoE by 13.6% on BOLA and 22.7% on BBA with almost the same average buffer occupancy. The only exception is RobustMPC: it performs almost equal to DeepBuffer, obtaining the Top-3 scheme in medium-network paths. Meanwhile, through analyzing the detailed results plotted in Figure 6(b) and Figure 6(c), we find that DeepBuffer with all considered weights can reach higher bitrate but fewer bitrate changes compared to baselines, while they work slightly worse in terms of stalling ratio, with the relative degradation of 0.07% (ω=0.1)-0.2% (ω=0.9) compared with Comyco.

Fast-network paths. Figure 7 shows the results from performing ABR algorithms on the 5G dataset. Here we illustrate the *huge* data waste of prior work in Figure 7(a). We reason that in the current network scenario, the bandwidth is sufficient for ABRs to pick chunks with the highest bitrate, while such schemes lack buffer control strategies like DeepBuffer, immediately downloading the chunk once the previous download process ends. In contrast, DeepBuffer picks the proper bitrate with preserving current buffer occupancy via buffer action (§IV-A), significantly reducing the average buffer size by 90.7% – an impressive number.

C. DeepBuffer vs. wastage-based schemes

We vary a set of buffer weights ω, sweeping from 0 to 1, to investigate the impact that each has on QoE_{lin} and buffer size. The wastage-based ABR algorithm includes Pensieve-ω, RobustMPC-ω, and PSWA. Results show DeepBuffer's outstanding generalization ability. Figure 5(d) indicates that DeepBuffer outperforms other schemes on slow-network paths since it reaches the highest QoE with the same buffer size. In turn, we see that Pensieve-ω fails to handle such diverse requirements on the buffer weight. That's because Pensieve-ω only adjusts the maximum buffer action solely, and it doesn't jointly consider the comprehensive effect brought by buffer and bitrate action. Same observation can be resulted in medium-path (Figure 6(d) and fast-network paths (Figure 7(d)) as well. Furthermore, RobustMPC-ω does consider both bitrate selection and buffer adjustment. It behaves better than Pensieve-ω in slow and medium-network paths, but its performance degrades heavily when the throughput predictions are incorrect in fast-path network scenario (Figure 7(d), [50]). In particular, comparing DeepBuffer with PSWA, we observe that DeepBuffer can generalize to different network environments, with the improvements on average QoE values within 17.4%-31.3% on slow-network paths, 6.5%-7.0% on medium-network paths, and 5.1% on fast-network paths respectively. In the meantime, compared with PSWA, DeepBuffer starts saving the buffer size when ω=0.4 on the slow-network paths. We reason that PSWA controls the maximum buffer by only considering average throughput, which not only lacks the feature selection but also neglects the influence on bitrate actions.

D. DeepBuffer with different bitrate ladders

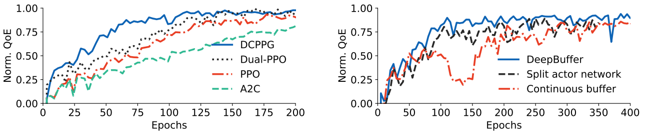
To validate the generalization of DeepBuffer, we conduct an experiment to test ABR schemes with two kinds of videos encoded by various bitrate ladder settings and report the main results in Table I. The selected video sets are Tears of Steel (ToS) [44], a short movie, and WA DA DA [33], a music video. We take DeepBuffer with ω=0, 0.5, and 0.7 for comparison. We don't compare Comyco and Pensieve in this part since they are not naturally designed to vary such multiple videos. Different from the previous experiment, we adopt O.46 score [47], which is the media session quality score in QoE_{itu} [51], to evaluate QoE. Previous work demonstrates that compared with QoE_{lin}, QoE_{itu} can better reflect the subjective quality evaluation of ABR.

As expected, DeepBuffer maintains good QoE_{itu} on two different kinds of videos. Among them, DeepBuffer-0, the scheme without considering the buffer sizes, ranks first in three network-video pairs, as it performs only 0.5% less than the best scheme on average QoE_{itu} in the ToS-Oboe scenario. Meanwhile, DeepBuffer-0.5 can balance the QoE_{itu}

TABLE I

Results of DeepBuffer with different videos on FCC and Oboe dataset. QoE metrics are computed as QoE$_{itu}$ [47].

1st		FCC [45]		Oboe [38]			FCC [45]		Oboe [38]	
2nd		QoE$_{itu}$ (↑)	Buffer (s) (↓)	QoE$_{itu}$ (↑)	Buffer (s) (↓)		QoE$_{itu}$ (↑)	Buffer (s) (↓)	QoE$_{itu}$ (↑)	Buffer (s) (↓)
	BOLA	2.62±0.78	21.73±10.31	3.53±0.76	38.23±17.82		3.14±0.67	18.83±5.07	3.82±0.69	23.65±5.59
	RMPC	2.84±0.61	18.32±10.93	3.68±0.71	33.56±19.79		3.31±0.57	23.04±9.72	3.93±0.62	21.98±8.83
	Fugu	2.88±0.63	17.46±12.41	3.68±0.70	34.11±20.86	WA	3.25±0.58	25.33±14.18	3.89±0.63	24.79±10.68
ToS	PSWA	2.64±0.48	10.89±3.40	3.40±0.71	10.30±4.02	DA	3.10±0.49	14.12±4.62	3.67±0.63	11.75±4.38
[44]	DB-0	2.89±0.58	22.83±5.92	3.66±0.68	25.75±7.50	DA	3.33±0.61	28.09±6.37	3.94±0.60	25.96±7.38
	DB-0.5	2.78±0.65	9.84±4.81	3.57±0.82	13.18±5.49	[33]	3.32±0.64	16.10±3.38	3.94±0.64	16.07±3.82
	DB-0.7	2.78±0.51	6.73±3.73	3.44±0.79	8.95±4.27		3.11±0.64	9.10±4.69	3.80±0.73	11.73±5.05



(a) "Sweeping" on-policy RLs (b) Action space design

Fig. 8. *Comparing DeepBuffer with other settings.*

performance and buffer occupancy over all considered scenarios. Especially compared with traditional ABR algorithms, it achieves the best QoE$_{itu}$ value while saving at least 36.7% on buffer size in the WA DA DA-Oboe scenario. Moreover, DeepBuffer-0.7 surpasses PSWA over all scenarios, with the improvements on average QoE$_{itu}$ up to 5%, as well as the significant decrease in average buffer occupancy up to 61.8%.

E. Ablation studies

Different RL methods. We compare DCPPG (§IV-B) with on-policy methods over a basic environment to prove its sample efficiency. Applying such a "complex" DRL method is not gilding the lily since Figure 8(a) shows the rapid learning efficiency of DCPPG. Technically, DCPPG rivals Dual-PPO [18], and it improves the normalized QoE by 5.46%-17.81% compared with PPO [52] and A2C [53].

Design of action space. Moreover, we compare different designs of the action space, such as combined bitrate and buffer space (i.e., $|A| \times |B|$), and continuous buffer actions. Figure 8(b) indicates that DeepBuffer with separated discrete action space gains the best performance on normalized QoE.

F. Real-world experiment

Finally, we conducted an experiment to validate how DeepBuffer performs in the wild. Specifically, we custom a new ABR rule on `Dash.js` [43] and play the video on Chrome V100. Note the client's buffer size can be easily adjusted by Dash.js API. The considered network scenarios cover 4G scenarios collected in Beijing Subway, public WiFi scenarios, and 5G scenarios. Figure 9 reports the average QoE$_{lin}$ value and buffer size for each scheme over different network conditions. Error bars span one standard deviation from the average. We reveal that DeepBuffer-0.1 outperforms existing schemes in terms of normalized QoE over all scenarios. In particular, DeepBuffer-0.1 improves QoE by 4.7%-55.5% in comparison to PSWA, RobustMPC, and Pensieve over metro networks. Meanwhile, it also reaches small buffer sizes, saving up to 9× in terms of average buffer. Moreover, Table II demystifies the influence of leveraging different kinds of congestion control algorithms (CCA) on DeepBuffer. Results are summarized as the normalized QoE (nQoE) and buffer size. As shown, AIMD-based schemes such as Reno and Cubic suffer from slow start effects, leading to a low estimated bandwidth. In contrast, pacing-based schemes like BBR performs much better than AIMD-based ones. Thus, we argue that available throughput is not independent of ABR algorithms. It can be further estimated by other critical features such as downloading chunks and CCAs [54]. We will discuss this interesting topic in the future.

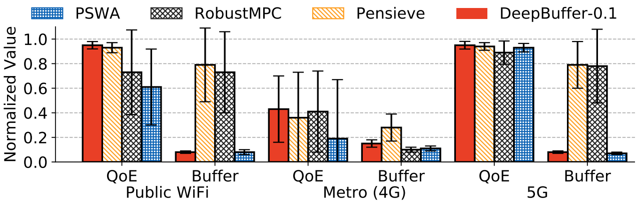


Fig. 9. *Comparing DeepBuffer with recent ABRs in the real-world.*

TABLE II

DeepBuffer meets congestion control algorithms

CCAs	Metric	Reno [55]	Cubic [56]	BBR [57]
DeepBuffer-0.1	nQoE/Buffer	0.52/0.15	0.57/0.10	0.81/0.08

VI. Conclusion

In this paper, we considered leveraging buffer-aware adaptive video streaming to overcome the increased data-wastage problem caused by sufficient bandwidth resources and limited bitrate improvements on videos. Modeling the task as multi-objective optimization, we proposed DeepBuffer, a DRL-based buffer-aware ABR scheme that considers varying multiple bitrate ladders, We have constructed DeepBuffer's training system, including its NN architectures and methodologies. Using a comprehensive trace-driven comparison of prior work and real-world deployment, we have illustrated that DeepBuffer can preserve the performance while reducing the buffer size by up to 90%, significantly restraining the data waste.

Acknowledgement We thank Nuowen Kan for his fruitful discussions and valuable feedback. This work was supported by NSFC under Grant 61936011, Beijing Key Lab of Networked Multimedia, and the Kuaishou-Tsinghua Joint Project.

References

[1] Sandvine, "2020 covid-19 phenomena spotlight report," 2020. [Online]. Available: https://www.sandvine.com/phenomena

[2] ——, "The global internet phenomena report january 2022," 2022. [Online]. Available: https://www.sandvine.com/phenomena

[3] A. Mondal, S. Sengupta, B. R. Reddy, M. Koundinya, C. Govindarajan, P. De, N. Ganguly, and S. Chakraborty, "Candid with youtube: Adaptive streaming behavior and implications on data consumption," in *Proceedings of the 27th Workshop on Network and Operating Systems Support for Digital Audio and Video*, 2017, pp. 19–24.

[4] T.-Y. Huang, C. Ekanadham, A. J. Berglund, and Z. Li, "Hindsight: Evaluate video bitrate adaptation at scale," in *Proceedings of the 10th ACM Multimedia Systems Conference*, 2019, pp. 86–97.

[5] H. Mao, S. Chen, D. Dimmery, S. Singh, D. Blaisdell, Y. Tian, M. Alizadeh, and E. Bakshy, "Real-world video adaptation with reinforcement learning," *arXiv preprint arXiv:2008.12858*, 2020.

[6] A. Bentaleb, B. Taani, A. C. Begen, C. Timmerer, and R. Zimmermann, "A survey on bitrate adaptation schemes for streaming media over http," *IEEE Communications Surveys & Tutorials*, 2018.

[7] Z. Li, X. Zhu, J. Gahm, R. Pan, H. Hu, A. C. Begen, and D. Oran, "Probe and adapt: Rate adaptation for http video streaming at scale," *IEEE JASC*, vol. 32, no. 4, pp. 719–733, 2014.

[8] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson, "A buffer-based approach to rate adaptation: Evidence from a large video streaming service," *SIGCOMM 2014*, vol. 44, no. 4, pp. 187–198, 2014.

[9] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman, "Bola: Near-optimal bitrate adaptation for online videos," *IEEE/ACM Transactions on Networking*, vol. 28, no. 4, pp. 1698–1711, 2020.

[10] H. Mao, R. Netravali, and M. Alizadeh, "Neural adaptive video streaming with pensieve," in *SIGCOMM 2017*, 2017.

[11] T. Huang, C. Zhou, R.-X. Zhang, C. Wu, X. Yao, and L. Sun, "Stick: A harmonious fusion of buffer-based and learning-based approach for adaptive streaming," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 1967–1976.

[12] T. Huang, C. Zhou, R.-X. Zhang *et al.*, "Comyco: Quality-aware adaptive video streaming via imitation learning," in *ACM Multimedia 2019*, 2019, pp. 429–437.

[13] F. Y. Yan, H. Ayers, C. Zhu, S. Fouladi, J. Hong, K. Zhang, P. Levis, and K. Winstein, "Learning in situ: a randomized experiment in video streaming," in *17th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 20)*, 2020, pp. 495–511.

[14] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli, "A control-theoretic approach for dynamic adaptive video streaming over http," in *SIGCOMM 2015*. ACM, 2015, pp. 325–338.

[15] P. K. Yadav, A. Shafiei, and W. T. Ooi, "Quetra: A queuing theory approach to dash rate adaptation," in *Proceedings of the 25th ACM international conference on Multimedia*, 2017, pp. 1130–1138.

[16] T. Huang, R. Zhang, and L. Sun, "Zwei: A self-play reinforcement learning framework for video transmission services," *IEEE Transactions on Multimedia*, 2021.

[17] K. Deb, "Multi-objective optimization," in *Search methodologies*. Springer, 2014, pp. 403–449.

[18] D. Ye, Z. Liu, M. Sun, B. Shi, P. Zhao, H. Wu, H. Yu, S. Yang, X. Wu, Q. Guo *et al.*, "Mastering complex control in moba games with deep reinforcement learning," *arXiv preprint arXiv:1912.09729*, 2019.

[19] K. Cobbe, J. Hilton, O. Klimov, and J. Schulman, "Phasic policy gradient," *arXiv preprint arXiv:2009.04416*, 2020.

[20] J. Jiang, V. Sekar, and H. Zhang, "Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive," *TON*, vol. 22, no. 1, pp. 326–340, 2014.

[21] X. Zuo, Y. Jiayu, M. Wang, and Y. Cui, "Adaptive bitrate with user-level qoe preference for video streaming," in *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*. IEEE, 2022, pp. 1–10.

[22] L. Plissonneau, E. Biersack, and P. Juluri, "Analyzing the impact of youtube delivery policies on user experience," in *2012 24th International Teletraffic Congress (ITC 24)*. IEEE, 2012, pp. 1–8.

[23] A. Finamore, M. Mellia, M. M. Munafo, R. Torres, and S. G. Rao, "Youtube everywhere: Impact of device and infrastructure synergies on user experience," in *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, 2011, pp. 345–360.

[24] G. Zhang, K. Liu, H. Hu, V. Aggarwal, and J. Y. Lee, "Post-streaming wastage analysis–a data wastage aware framework in mobile video streaming," *IEEE Transactions on Mobile Computing*, vol. 22, no. 1, pp. 389–401, 2021.

[25] H. Riiser, P. Vigmostad, C. Griwodz, and P. Halvorsen, "Commute path bandwidth traces from 3g networks: analysis and applications," in *Proceedings of the 4th ACM Multimedia Systems Conference*. ACM, 2013, pp. 114–118.

[26] A. Narayanan, E. Ramadan, R. Mehta *et al.*, "Lumos5g: Mapping and predicting commercial mmwave 5g throughput," in *IMC 20*, New York, NY, USA, 2020.

[27] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the h. 264/avc video coding standard," *IEEE Transactions on circuits and systems for video technology*, vol. 13, no. 7, pp. 560–576, 2003.

[28] G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand, "Overview of the high efficiency video coding (hevc) standard," *TSCVT*, 2012.

[29] B. Bross, Y.-K. Wang, Y. Ye, S. Liu, J. Chen, G. J. Sullivan, and J.-R. Ohm, "Overview of the versatile video coding (vvc) standard and its applications," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 31, no. 10, pp. 3736–3764, 2021.

[30] J. Bankoski, P. Wilkins, and Y. Xu, "Technical overview of vp8, an open source video codec for the web," in *2011 IEEE International Conference on Multimedia and Expo*. IEEE, 2011, pp. 1–6.

[31] D. Mukherjee, J. Bankoski, A. Grange, J. Han, J. Koleszar, P. Wilkins, Y. Xu, and R. Bultje, "The latest open-source video codec vp9-an overview and preliminary results," in *2013 Picture Coding Symposium (PCS)*. IEEE, 2013, pp. 390–393.

[32] Y. Chen, D. Murherjee, J. Han, A. Grange, Y. Xu, Z. Liu, S. Parker, C. Chen, H. Su, U. Joshi *et al.*, "An overview of core coding tools in the av1 video codec," in *2018 Picture Coding Symposium (PCS)*. IEEE, 2018, pp. 41–45.

[33] Kep1er, "Wa da da," https://www.youtube.com/watch?v=n0j5NPptyM0, 2022.

[34] A. Francis, "4k video at sd bitrates with av1," https://bitmovin.com/av1-4k-video-sd-bitrates/, 2022.

[35] T.-Y. Huang, N. Handigol, B. Heller, N. McKeown, and R. Johari, "Confused, timid, and unstable: picking a video streaming rate is hard," in *IMC 2012*, 2012, pp. 225–238.

[36] K. Technology, "Announcement of the results for the year ended december 31, 2021," https://www.kuaishou.com, 2022.

[37] "Enviviodash3," https://dash.akamaized.net/envivio/EnvivioDash3/, 2016.

[38] Z. Akhtar, Y. S. Nam, R. Govindan *et al.*, "Oboe: auto-tuning video abr algorithms to network conditions," in *SIGCOMM 2018*, 2018.

[39] S. Huang and S. Ontañón, "A closer look at invalid action masking in policy gradient algorithms," *arXiv preprint arXiv:2006.14171*, 2020.

[40] J. Li, S. Koyamada, Q. Ye, G. Liu, C. Wang, R. Yang, L. Zhao, T. Qin, T.-Y. Liu, and H.-W. Hon, "Suphx: Mastering mahjong with deep reinforcement learning," *arXiv preprint arXiv:2003.13590*, 2020.

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

[42] H. Mao, P. Negi, A. Narayan, H. Wang, J. Yang, H. Wang, R. Marcus, R. Addanki, M. Khani, S. He *et al.*, "Park: An open platform for learning augmented computer systems," in *NIPS 2019*, 2019.

[43] DASH, "Dash," 2019. [Online]. Available: https://dashif.org/

[44] O. Movie, "Tears of steel," https://mango.blender.org/, 2013.

[45] M. F. B. Report, "Raw data measuring broadband america 2016," https://www.fcc.gov/, 2016, [Online; accessed 19-July-2016].

[46] Z. Meng, Y. Guo, Y. Shen, J. Chen, C. Zhou, M. Wang, J. Zhang, M. Xu, C. Sun, and H. Hu, "Practically deploying heavyweight adaptive bitrate algorithms with teacher-student learning," *IEEE/ACM Transactions on Networking*, vol. 29, no. 2, pp. 723–736, 2021.

[47] W. Robitza, S. Göring, A. Raake, D. Lindegren, G. Heikkilä, J. Gustafsson, P. List, B. Feiten, U. Wüstenhagen, M.-N. Garcia, K. Yamagishi, and S. Broom, "HTTP Adaptive Streaming QoE Estimation with ITU-T Rec. P.1203 – Open Databases and Software," in *9th ACM Multimedia Systems Conference*, Amsterdam, 2018.

[48] M. Laumanns, L. Thiele, and E. Zitzler, "An adaptive scheme to generate the pareto front based on the epsilon-constraint method," in *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2005.

[49] 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.

[50] A. Narayanan, X. Zhang, R. Zhu, A. Hassan, S. Jin, X. Zhu, X. Zhang, D. Rybkin, Z. Yang, Z. M. Mao *et al.*, "A variegated look at 5g in the wild: performance, power, and qoe implications," in *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, 2021, pp. 610–625.

[51] O. ITU, "Series p: Telephone transmission quality, telephone installations, local line networks methods for objective and subjective assessment of quality."

[52] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[53] 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.

[54] A. Alomar, P. Hamadanian, A. Nasr-Esfahany, A. Agarwal, M. Alizadeh, and D. Shah, "Causalsim: Toward a causal data-driven simulator for network protocols," *arXiv preprint arXiv:2201.01811*, 2022.

[55] G. R. Wright and W. R. Stevens, *TCP/IP Illustrated, Volume 2 (paperback): The Implementation*. Addison-Wesley Professional, 1995.

[56] 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.

[57] N. Cardwell, Y. Cheng, S. H. Yeganeh, I. Swett, V. Vasiliev, P. Jha, Y. Seung, M. Mathis, and V. Jacobson, "Bbrv2: A model-based congestion control," in *Presentation in ICCRG at IETF 104th meeting*, 2019.