Grad: Learning for Overhead-aware Adaptive Video Streaming with Scalable Video Coding

Technical Report

Yunzhuo Liu
Shanghai Jiao Tong University
liu445126256@sjtu.edu.cn

Bo Jiang*
Shanghai Jiao Tong University
bjiang@sjtu.edu.cn

Tian Guo
Worcester Polytechnic Institute
tian@wpi.edu

Ramesh K. Sitaraman
University of Massachusetts Amherst
& Akamai Technologies
ramesh@cs.umass.edu

Don Towsley
University of Massachusetts Amherst
towsley@cs.umass.edu

Xinbing Wang
Shanghai Jiao Tong University
xwang8@sjtu.edu.cn

ABSTRACT

Video streaming commonly uses Dynamic Adaptive Streaming over HTTP (DASH) to deliver good Quality of Experience (QoE) to users. Videos used in DASH are predominantly encoded by single-layered video coding such as H.264/AVC. In comparison, multi-layered video coding such as H.264/SVC provides more flexibility for upgrading the quality of buffered video segments and has the potential to further improve QoE. However, there are two challenges for using SVC in DASH: *(i)* the complexity in designing ABR algorithms; and *(ii)* the negative impact of SVC's coding overhead. In this work, we propose a deep reinforcement learning method called Grad for designing ABR algorithms that take advantage of the quality upgrade mechanism of SVC. Additionally, we quantify the impact of coding overhead on the achievable QoE of SVC in DASH, and propose jump-enabled hybrid coding (HYBJ) to mitigate the impact. Through emulation, we demonstrate that Grad-HYBJ, an ABR algorithm for HYBJ learned by Grad, outperforms the best performing state-of-the-art ABR algorithm by 17% in QoE.

KEYWORDS

Scalable video coding; adaptive bitrate algorithm; reinforcement learning

1 INTRODUCTION

Video streaming over the Internet has grown rapidly over the past years, and it is predicted to contribute 82% of the total IP traffic in 2022 [3]. The growth is accompanied by increasing user demands on better Quality of Experience (QoE), which has been shown to have a huge impact on content providers' revenue [12]. Achieving high QoE is challenging as it often involves taking into account conflicting requirements such as minimal rebuffering and high bitrates in the presence of network variability.

A technology widely used to cope with network variability is Dynamic Adaptive Streaming over HTTP (DASH). In DASH, videos are divided into small segments, each encoded at several different quality levels. Adaptive bitrate (ABR) algorithms are then used to decide dynamically the quality level of each segment, based on information such as the playback buffer state and the estimated network bandwidth. ABR algorithms aim to maximize the overall QoE by striking a balance between multiple conflicting goals such as *high quality*, *minimal rebuffering* and *few quality switches* [18, 22, 35]. A lot of work on internet video streaming has been devoted to the design of better ABR algorithms [9–11, 18, 22, 32, 35], and further improvements are still desired.

Most ABR algorithms are designed to work with the video coding scheme called Advanced Video Coding (AVC). Versions of the same video segment at different quality levels are encoded independently of each other, and segments are downloaded in their playback order. Typically all decisions on segment qualities are *final*, i.e., ARB algorithms only execute one download for each segment. Consequently, the ability of ABR algorithms to maximize QoE depends critically on the accuracy of its predictions, either explicit or implicit, for the relatively long-term average bandwidth. However, such predictions are often inaccurate [16, 18], so an ABR algorithm may fail to strike the right balance between different QoE goals. For instance, it may download a lot of low quality segments before realizing that a higher quality could have been selected. On the other hand, if the selected quality is too high to be sustainable, it will then result in rebuffering. The problems are exacerbated by the fact that very frequent quality switching degrades user QoE.

To address the aforementioned limitations of AVC, we investigate in this work the problem of *improving user QoE with the alternative coding scheme Scalable Video Coding (SVC).* In contrast to AVC, SVC encodes different versions of the same segment in an incremental manner. High quality versions can be obtained from lower quality ones by adding their difference. This allows ABR algorithms to be conservative and download low quality segments to avoid rebuffering in the presence of bandwidth uncertainty. If the bandwidth turns out to be high, the segments can then be upgraded to higher qualities. The ability to upgrade provides ABR algorithms with more flexibility in making download decisions and hence helps even out bandwidth fluctuations and improve QoE.

However, the use of SVC in DASH faces two challenges. The *first* one is the enlarged decision space due to the quality upgrade mechanism. An ABR algorithm not only needs to select quality levels for

*Corresponding author.

This work was supported in part by National Key R&D Program of China 2018AAA0101200, National Natural Science Foundation of China under Grant Numbers 61532012, 61960206002 and 61829201, Science and Technology Innovation Program of Shanghai (Grant 18XD1401800), Shanghai Key Laboratory of Scalable Computing and Systems, and the U.S. National Science Foundation under Grant Numbers CNS-1755659, CNS-1815619, CNS-1763617, CNS-1901137, and CNS-1617437.

new segments, but also has to decide whether to upgrade buffered segments and to which quality levels. This adds to the complexity in designing ABR algorithms. As a result, existing ABR algorithms often underutilize the flexibility provided by SVC, either limiting upgrades to the most recently downloaded segment [13, 34], or using unoptimized handcrafted rules [5, 8, 16, 22, 25]. The *second* challenge is the coding overhead. SVC typically requires more bits than AVC to achieve the same visual quality. This coding overhead consumes extra bandwidth and can potentially degrade QoE. However, most existing ABR algorithms for SVC do not explicitly mitigate the negative impact of overhead on QoE [5, 8, 13, 16, 25, 34], resulting in suboptimal performance.

We address the challenges associated with SVC by answering the following questions: *(i)* How to design ABR algorithms that better utilize the built-in quality upgrade mechanism? *(ii)* What is the impact of coding overhead on QoE and how to mitigate it? We make the following main contributions.

- We propose Grad, a Deep Reinforcement Learning (DRL) method for designing ABR algorithms that fully utilize the quality upgrade mechanism of SVC. We tailor actions for more effective learning that leads to better upgrade policies.
- We propose *jump-enabled hybrid coding* (HYBJ), an overhead-aware way of using SVC in DASH. This design is grounded on our empirical evaluations of the impact of coding overhead on achievable SVC QoE. In particular, we find that SVC starts to perform worse than AVC when coding overhead per enhancement layer exceeds 7%.
- Using Grad and HYBJ, we obtain an ABR algorithm called Grad-HYBJ that achieves a 17.0% higher QoE with only 2.2% more transmitted data, compared to the best performing state-of-the-art ABR algorithm. Grad-HYBJ transmits 12.7% more data, but achieves 25.9% higher QoE compared to the most bandwidth-efficient method.

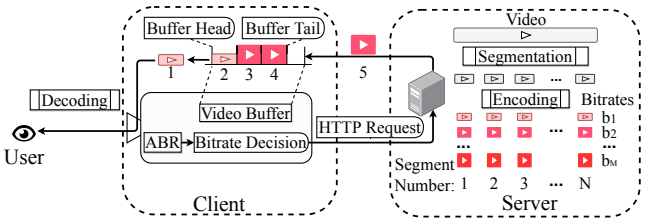


Figure 1: *DASH.*

2 BACKGROUND & RELATED WORK

2.1 Dynamic Adaptive Streaming over HTTP

Dynamic Adaptive Streaming over HTTP (DASH) is a standard that allows the client player, e.g. a web browser, to adaptively stream segmented video from HTTP servers based on network conditions. Figure 1 illustrates the key components of a DASH architecture. At the core of DASH are *video segments* and *Adaptive Bitrate (ABR) algorithms*. Each video segment corresponds to a short interval of playback time. For example, a minute-long video might be divided to 5 segments of 12 seconds each. Each segment can have different quality versions, i.e., *bitrates*, associated with it. Those versions can be generated with different video coding methods. In this work we consider two coding methods as described in the next subsection. During playback, the client player will leverage an ABR algorithm for bitrate adaptation, described in Section 2.3, to decide which quality version to download for each video segment. The download decisions are often made in the same order as the video segment playback order. The player stores and plays the downloaded video segments using a first-in-first-out queue, often referred to as *buffer*. The video segments that have waited in the queue for the longest and shortest time are referred to as the *buffer head* and *buffer tail*, respectively. When the buffer is full, ABR algorithms typically pause downloading the next segment until the buffer head is played back.

2.2 Video Coding

Video coding standards such as Advanced Video Coding (AVC) [33] are used to compress raw videos. AVC is commonly used in Video on Demand (VoD) with DASH to encode one video segment to independent versions at different quality levels. In contrast, a standard called Scalable Video Coding (SVC) [28] can implement those different versions as *dependent* video layers. Video segment of a given quality can be reconstructed with a *base layer* (BL) and one or more *enhancement layers* (ELs). For example, to support m levels of quality, one can encode a video segment to a base layer and $m-1$ enhancement layers. We call this *vanilla SVC*. The multi-layer property of SVC provides better flexibility for bitrate adaption as ABR algorithms can adjust prior bitrate decisions by *upgrading* buffered video segments with newly downloaded enhancement layers. However, an SVC-encoded video segment is often larger in size than its AVC-encoded counterpart of the same quality level. Such size differences are referred to as *coding overhead* and they are usually proportional to the number of enhancement layers of the desired quality level [14, 17].

To reduce the coding overhead associated with SVC in DASH, prior work proposed *hybrid coding* [14, 22]. The idea is to reduce the number of required enhancement layers for each quality level by keeping more base layers of higher qualities. In this work, we propose a *jump-enabled hybrid coding* that focuses on further reducing the number of enhancement layers by allowing the quality of a segment to jump multiple levels using only one enhancement layer. Basing on our hybrid coding, we further learn an overhead-aware ABR algorithm, as described in Section 3.3.

2.3 ABR Algorithm

In this work, we consider ABR algorithms that target AVC and SVC-encoded videos, respectively. ABR algorithm designed for AVC has garnered a lot of interests as AVC is commonly used in DASH. Prior work [15, 18, 32, 35] improves video streaming quality with approaches ranging from control-theoretical approach to neural networks. Most existing AVC-based ABR algorithms were designed without quality upgrade mechanism and they only decide the quality level for the next video segment. Recently, BOLA-FAST [31] considered quality upgrade for AVC-encoded videos by replacing buffered segments with new ones of higher quality. However, such video segment replacement incurs non-trivial bandwidth costs and is rarely supported by other AVC-based ABR algorithms.

Existing SVC-based ABR algorithms [5, 8, 13, 16, 25, 34] were designed with quality upgrade mechanism to exploit the flexibility of SVC-encoded video segments. One of the key challenges is to handle the larger decision space associated with it. Prior work [13, 34] addressed this by limiting the actions to upgrading the buffer tail or downloading base layer for the next segment, leaving the flexibility underutilized. Handcrafted rules have also been proposed to exploit the flexibility [5, 8, 16, 25] by allowing more actions, but they do not balance well across different QoE goals.

Our work differs from prior work by designing ABR algorithms with quality upgrade mechanism that can operate on any buffered segment through a learning-based approach. It automatically balances between different QoE goals and forms quality upgrade policies that can benefit the overall QoE.

2.4 Reinforcement Learning

Reinforcement Learning (RL) is a type of machine learning technique that is commonly used for guiding an agent to maximize the *cumulative reward* through a sequence of actions. In RL, agents learn their policies, defined as the probability distribution of actions in each state, by interacting with the environment.

A number of prior work leveraged RL for developing ABR algorithms [9–11, 18] that target AVC-encoded videos. The ABR algorithm is modeled as the agent that makes decisions such as which video segment to download next, in order to maximize a predefined optimization metric. By using reinforcement learning, ABR policies that specify which action to take in each state, can be obtained automatically to form the final ABR algorithm. Notably, Pensieve pioneered the use of deep reinforcement learning for generating optimized ABR algorithms. Specifically, Pensieve used a popular actor-critic architecture [19] consisting of two neural networks. The *actor network* gives the policy by mapping states to probability distributions of actions, while the *critic network* evaluates the policy by predicting its expected total reward.

Our work also explored the use of deep reinforcement learning for generating ABR algorithms, but with the focus on a more challenging action space introduced by quality upgrade mechanism.

3 LEARNING ABR ALGORITHMS WITH QUALITY UPGRADE MECHANISM

3.1 Overview

We describe Grad, a Deep Reinforcement Learning (DRL) method, for designing ABR algorithms that utilize the quality upgrade mechanism of SVC. Grad can also be used to learn ABR algorithms for AVC, via configuring the appropriate quality upgrade related overhead. The learning agent automatically balances the cost and gain of quality upgrade and learns overhead-aware ABR policies. Unlike traditional AVC-oriented ABR algorithms without quality upgrade mechanism, our learning agent needs to explore a larger decision space: *(i)* choosing a quality version for the next video segment, *(ii)* or upgrading the quality of a buffered segment. One key challenge associated with the enlarged decision space is the difficulty in exploring and learning good quality upgrade policies. Below we first present an overview of how we design Grad, followed by the tailored action designs for both the vanilla SVC and our jump-aware hybrid coding in Sections 3.2 and 3.3, respectively.

Optimization metric. As described in Section 2.4, an RL agent learns policies by optimizing a cumulative reward. In this work, we use the following QoE metric to quantify the attained reward,

$$QoE = \sum_{n=1}^{N} \log\left(\frac{R_n}{R_{\min}}\right) - \log\left(\frac{R_{\max}}{R_{\min}}\right) \sum_{n=1}^{N} T_n - \sum_{n=1}^{N-1} |\log(R_{n+1}) - \log(R_n)| \times \frac{\max(R_{n+1}, R_n)}{\min(R_{n+1}, R_n)}, \quad (1)$$

where N is the total number of video segments, R_n and T_n are the bitrate and rebuffering time of the n-th segment, $R_{\min}$ and $R_{\max}$ are the bitrates of the lowest and highest segment quality, respectively. Note that we calculate the bitrate for an SVC-encoded segment based on its AVC-encoded counterpart of the same visual quality. In other words, we do not account for the extra bits associated with the coding overhead of SVC in bitrate calculation.

The above QoE metric follows a commonly used general formula: $QoE = \mu \sum_{n=1}^{N} f(R_n) - \nu \sum_{n=1}^{N} T_n - \xi \sum_{n=1}^{N-1} |f(R_{n+1}) - f(R_n)|$. As in prior work [18, 22, 35], we let $f(\cdot)$ be the binary logarithm, parameter μ be 1, and ν be the highest segment quality. We set ξ equal to the quotient between the bitrates of two consecutive segments, which penalizes steeper quality changes more [20].

Note that this QoE metric considers the following three important and often conflicting aspects, *segment quality*, *rebuffering time*, and *video smoothness*. *Segment quality*, referring to the visual quality of a single segment, is mostly determined by the segment bitrate and is accounted for by the first term in Eq. (1). It is also referred to as *bitrate utility*. *Rebuffering time* denotes the time delay after all previously downloaded segments have been played back and before the new segment is ready to be played, and is accounted for by the second term. Finally, *video smoothness*, which quantifies the effect of segment quality switches, is considered in the third term.

Network input. We chose the following state inputs $s_t = (b_t, e_t, z_t, \boldsymbol{x}_t, \boldsymbol{d}_t, \boldsymbol{q}_t, \boldsymbol{w}_t)$ for training the RL agent with SVC-encoded videos. In particular, b_t is the fraction of the buffer that is currently occupied by downloaded segments; e_t is the number of segments in the video that have not been downloaded; z_t denotes the data size of the ELs specified by our actions; $\boldsymbol{x}_t$ and $\boldsymbol{d}_t$ represent the measured throughput and time over the past n downloads and we used $n = 8$ in this work; $\boldsymbol{q}_t$ denotes the quality of each video segment in the buffer; $\boldsymbol{w}_t$ defines the time before each buffered video segment is played back to user. Four of the inputs $(b_t, e_t, \boldsymbol{x}_t, \boldsymbol{d}_t)$ were also used by Pensieve [18], while the remaining inputs $(z_t, \boldsymbol{q}_t, \boldsymbol{w}_t)$ account for the enlarged decision space associated with SVC.

Network architecture. Our *actor-critic* network (Figure 2) consists of an input layer, three hidden layers, and an output layer. The input layer consists of one-dimension CNN layers that process $(\boldsymbol{x}_t, \boldsymbol{d}_t, \boldsymbol{q}_t, \boldsymbol{w}_t)$, and fully connected layers that process (b_t, e_t, z_t). The output of the *actor network* is a vector specifying the probability distribution of actions, while the output of the *critic network* is a value predicting the cumulative reward.

Policy gradient. The *actor-critic* algorithm trains the networks using policy gradient. The bitrate decision at step t generates a reward r_t, and policy gradient aims to increase the cumulative reward $\sum_{t=0}^{\infty} \gamma^t r_t$, where γ is a discount factor. The gradient can

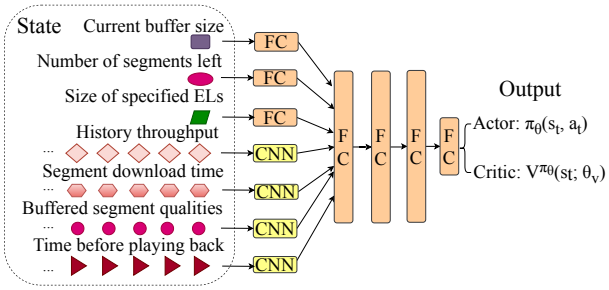
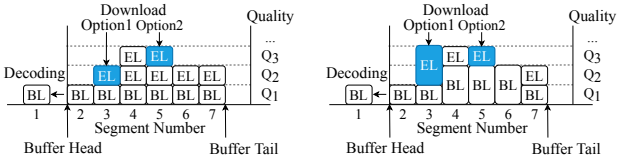


Figure 2: *Network architecture of Grad.*



(a) Vanilla SVC: approach-neighbour. (b) Hybrid coding: match-neighbour.

Figure 3: *Neighbor-related actions for vanilla SVC and hybrid coding.*

be computed as:

$$\nabla E_{\pi_\theta}\left[\sum_{t=0}^{\infty}\gamma^t r_t\right] = E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) A^{\pi_\theta}(s,a)] \tag{2}$$

The policy parameter θ is the parameter of the actor network. $\pi_\theta(s,a)$ is the output probability for action a in state s. $A^{\pi_\theta}(s,a)$ is the *advantage function* given by the difference between the expected reward of taking the deterministic action a in state s and the expected average reward following policy π_θ. During training, we sample a trajectory of actions to compute $A(s_t, a_t)$ as an unbiased estimation of $A^{\pi_\theta}(s_t, a_t)$ using the *temporal difference* method:

$$A(s_t, a_t) = r_t + \gamma V^{\pi_\theta}(s_{t+1}; \theta_v) - V^{\pi_\theta}(s_t; \theta_v) \tag{3}$$

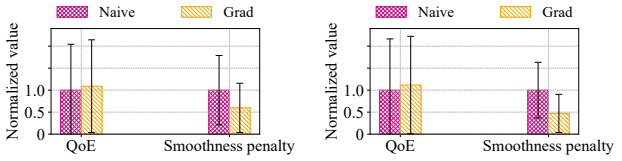
θ_v is the parameter of the critic. The critic outputs $V^{\pi_\theta}(s_t; \theta_v)$ as an estimation of the value function $v^{\pi_\theta}(s_t)$ that represents the accumulative reward from input state s_t following actor policy π_θ. Actor network parameter θ is updated using the following equation:

$$\theta \leftarrow \theta + \alpha_a \sum_t \nabla_\theta \log \pi_\theta(s,a) A(s_t, a_t) + \beta \nabla_\theta H(\pi_\theta(\cdot|s_t)) \tag{4}$$

The second part $\beta \nabla_\theta H(\pi_\theta(\cdot|s_t))$ encourages the actor to explore different policies. $H(\cdot)$ is the entropy. β and α_a are the exploration and learning rate of the actor. For the critic, the update of its parameters is as follows:

$$\theta_v \leftarrow \theta_v - \alpha_v \sum_t \nabla_{\theta_v} (A(s_t, a_t))^2 \tag{5}$$

where α_v is its learning rate. In the training, we configured α_a and α_v to be 0.0001, γ to be 0.99, and β to decay linearly from 3.0 to 0.05 over 50000 iterations.



(a) Effectiveness of *approach-neighbor.* (b) Effectiveness of *match-neighbor.*

Figure 4: *Comparison between our action designs and naive designs without the neighbor-related actions.* *Note that Naive-HYBJ simply allows upgrading the quality of any buffered segment to any available level.*

3.2 Action Design for Vanilla SVC

The first type of action, referred to as *download-base*, simply downloads the base layer for the next segment. This is akin to downloading the lowest bitrate quality of AVC-encoded videos.

The second type of action, referred to as *upgrade-by-one*, downloads the next enhancement layer to upgrade the quality of a buffered video segment by one level. For example, if the chosen video segment is currently at quality level w, this action will choose the w-th enhancement layer which upgrades the quality of the segment to $w+1$ where $w+1 \leq M$.

With only the above two action types, it might become difficult for the RL agent to explore a policy that achieves good video smoothness. Recall that video smoothness is an important QoE metric that desires consecutive video segments to be at the same quality level. The key challenge stems from the *enlarged decision space* associated with quality upgrade mechanism. With the enlarged decision space, the next download decision can affect the quality of any buffered segment. Thus its impact on video smoothness depends on the state of the entire buffer. As the state space grows exponentially with the buffer size and most upgrade actions are likely to damage video smoothness, e.g., when one of its neighbors already has lower quality, the RL agent has difficulty in exploring upgrade policies that can maintain good video smoothness after executing a series of actions of downloading ELs with different buffer states.

To help the RL agent learn good quality upgrade policies, we added a third type of action called *approach-neighbour*. This action type is a subset of the second one and is designed to help the agent learn smoothness-friendly quality upgrade policies by explicitly specifying better candidate segments. The agent will download the w-th enhancement layer for a buffered video segment at quality level w if at least one of its neighboring segments has a higher quality. If multiple video segments in the buffer satisfy the neighbor condition, the RL agent can pick the video segment that is the closest to either buffer head or buffer tail. For example, in Figure 3(a), both video segments 3 and 5 (coded in blue) are candidates for quality upgrade. Figure 4(a) demonstrates that *approach-neighbor* action led to better QoE and less damage in video smoothness (detailed methodology in Section 4.1).

3.3 Action Design for Hybrid Coding

Before describing a set of actions tailored for hybrid coding, we first provide our rationale and our *jump-enabled hybrid coding*.

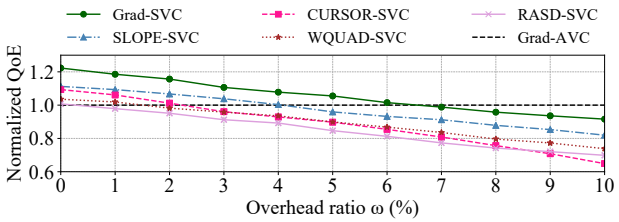


Figure 5: *Impact of coding overhead.* *QoEs were obtained with ω increasing from 1% to 10% in simulation, normalized to* Grad-AVC.

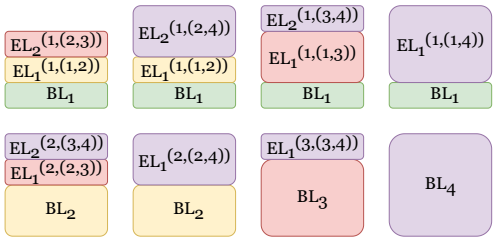


Figure 6: *Our proposed* jump-enabled *hybrid coding for SVC.* *The color of a layer here denotes the attained quality.*

Vanilla SVC incurs high coding overhead that can neutralize its benefits in DASH. It implements a video segment of the m-th quality level as a base layer and $m-1$ enhancement layers. Its coding overhead can be represented as $v(k) \cdot R_m$, where R_m is the average bitrate of an AVC-encoded video at quality level m, and $v(\cdot)$ is a function of the number of enhancement layers k that make up the SVC-encoded segment at quality level m. In this case, $k = m-1$. It is commonly agreed that $v(\cdot)$ increases monotonically with k and prior work suggested that $v(k) = k * \omega$, where ω can be set to 0.1 or 0.15 [14, 17]. Note that SVC base layers do not incur any overhead; in fact, they can be made compatible with AVC. Figure 5 shows the QoEs obtained from simulation with our Grad-SVC and four existing SVC-based ABR algorithms (SLOPE [5], CURSOR [8], WQUAD [16], and RASD [25]) under different ω, compared to the ABR algorithm learned for AVC using Grad (denoted as Grad-AVC, detailed methodology in Section 4.1). When the overhead ratio ω is small, e.g., smaller than 7%, at least one SVC-based ABR algorithm (i.e., our Grad-SVC) outperformed Grad-AVC. However, as the coding overhead continued to increase to 10%, all SVC-based algorithms including Grad-SVC had at least 9.2% lower QoE than Grad-AVC. Our observations suggest the potential of leveraging SVC to improve the QoE, when the coding overhead is low. However, as prior work [17, 21] and our investigation suggested (Table 1), using SVC would incur coding overhead that has $\omega > 10\%$.

Our proposed jump-enabled hybrid coding. Hybrid coding can be broadly thought of as a way of using SVC that mitigates the coding overhead associated with enhancement layers. For example, prior work [14] encoded each video segment first into multiple base layers at different qualities using AVC; enhancement layers were then generated for each base layer to produce more quality levels. We propose a new hybrid coding, referred to as *jump-enabled hybrid coding* (HYBJ), that generates enhancement layers for all possible quality upgrade combinations in advance. Formally, we denote the base layer at quality level m as BL_m and the i-th enhancement layer that upgrades the segment quality from level w to r as $EL_i^{(m,(w,r))}$, where $w < r \leq M$ and M is the total number of supported quality levels. Obviously HYBJ requires much more storage than vanilla SVC, but this problem can be largely mitigated (Section 4.6). We denote by l the maximum number of enhancement layers that can pile on a base layer. Note that $1 \leq i \leq l, 1 \leq l \leq M-1$. HYBJ with larger l provides more chances for quality upgrade and has more potential to benefit bitrate adaptation, but also *(i)* has higher storage cost, *(ii)* expands the exploration space of the agent and thus adds to the difficulty of learning, and *(iii)* includes enhancement layers with larger index i that introduce high overhead that outweighs the gain. Therefore, the choice of l is a tradeoff. In our implementation, we use $l = 2$.

Figure 6 illustrates all the layers we need to implement HYBJ with $M = 4$ and $l = 2$. The key is to allow the quality of a segment to jump to any available quality level using only one enhancement layer. By doing so we limit the overhead incurred by quality upgrade to the coding overhead associated with *only one* enhancement layer. We describe and compare to a different hybrid coding [22] in Section 4.

Actions for hybrid coding. Actions for hybrid coding are similar to those of vanilla SVC but with important modifications. The first action, *download-base*, downloads a base layer for the next segment of a chosen quality instead of only the lowest quality. The second action, *upgrade-by-one*, is the same action as the one for vanilla SVC. The third action *match-neighbor* differs from *approach-neighbor* for vanilla SVC in that the downloaded EL will upgrade the segment quality to the same as its neighbor, as illustrated in Figure 3(b). If both neighbors have higher qualities, the RL agent will refer to the quality of the right neighbor. The function of the second action here is to account for scenarios where all video segments are of the same quality and the third action cannot find a target segment. We limit the upgrade in the second action to one level based on the intuition that the quality adjustment requirement is low when all video segments in the buffer have the same quality. Similar to vanilla SVC, we also observed the effectiveness of *match-neighbor* action for hybrid coding in Figure 4(b). We also applied this action design to learning ABR algorithms for AVC-encoded videos.

4 EVALUATION

In this section we evaluate Grad with HYBJ (Grad-HYBJ) by comparing it with state-of-the-art ABR algorithms and different coding methods, including a *progressive hybrid coding* (HYBP). Performance is evaluated in terms of QoE-related metrics (Section 4.3), bandwidth and storage costs (Section 4.6), and reaction time (Section 4.7).

4.1 Methodology

Testbed setup. For training and evaluating Grad, we used a simulated and an emulated video streaming systems respectively. The simulation system, similar to that used in Pensieve [18], was for accelerating the training process and also used to study the impact of SVC coding overhead on obtained QoE (Figure 5). The emulator allowed us to conduct performance evaluation of video streaming in a controlled network environment. We implemented a video

server with *Nginx* Version 1.10 and a client player in *python* Version 3.6. We configured the round trip time to be 80 ms and used Mahimahi [23] to emulate the bandwidth between the video server and client by replaying network traces (described below). The client had a video buffer of 60 seconds and logged important streaming events, such as bitrate switches and rebuffering, for post analysis.

Videos. We used the following video statistics, adopted in prior work [4, 18], in the simulator. The video has a total duration of 192 seconds and consists of 4-second segments. For AVC encoding, each segment has six different bitrates, i.e.,*[300 Kbps, 750 Kbps, 1200 Kbps, 1850 Kbps, 2850 Kbps, 4300 Kbps]*, under Constant Bit Rate (CBR) mode. The simulator calculated the size of each AVC segment basing on its bitrate and duration and used the segment size and network condition to simulate the downloading process. For segments whose encoding involves SVC, we added the coding overhead to the size of their AVC counterparts as discussed in Section 2.2. The configured overhead ratio fluctuated within a range following an uniform distribution. We configured the range to be consistent with the overhead level of current SVC technology for obtaining an overhead-aware ABR algorithm workable in practice.

To determine the range, we empirically measured the overhead of three-layer SVC using an open source software `JSVM` recommended by H.264/SVC [29] to encode four videos with different motion and texture details. 200 frames of each video were used. We used the following configurations of *JSVM*: *(i)* Group of Pictures (GoP) of 8; *(ii)* an intra period of 32; *(iii)* inter layer prediction mode 2; *(iv)* spatial scalability, with resolutions of 360P, 720P and 1080P. Note that though temporal and quality scalability usually incur smaller overhead[14], they cannot cover a wide range of bitrate choices, e.g. it is not reasonable to increase the bitrate of a 360P segment from 300 Kbps to 4300 Kbps for the limited improvement in visual quality. Thus we implemented our design with spatial scalability, and the proper use of other two scalability types in bitrate adaptation remains for further work; *(v)* fixed Quantization Parameter (QP) mode. The measurement is based on Bjøntegaard Delta-rate (BD-rate)[6], which quantifies the average difference in bitrates of encoded videos with the same quality. Peak Signal-to-Noise Ratio (PSNR) is used to quantify video quality. We used four QPs for base layer {16,20,24,28} as recommended, and used two QP offsets {0,+2}. As shown in Table 1, the ranges for $v(1)$, $v(2)$ are [10%, 20%] and [20%, 40%], respectively.

The emulator uses the first 192 seconds of *BigBuckBunny* video. We implemented the bitrates *[300 Kbps, 750 Kbps, 1200 Kbps, 1850 Kbps, 2850 Kbps, 4300 Kbps]* as *[144P, 240P, 360P, 480P, 720P, 1080P]*. To encode the video, we first split it into segments of 4 seconds and then used the fixed QP mode in *JSVM* to encode them into base layers (also served as AVC segments) at designated bitrates. Then we used the same QP parameters for each quality level to generate enhancements layers in our HYBJ.

Network Traces. We used traces from both 3G and 4G mobile network datasets [7, 26, 27]. Those traces were collected in different scenarios and contain per second throughput information for different durations. In particular, the 4G dataset [26] was collected under different mobility patterns including home, pedestrian, car, tram and train, and with network throughput ranging from 0 to 173 Mbps over 2100 minutes. Additionally, the two 3G network datasets were collected in Norway with different means of transportation [27] and in Sydney [7], respectively. The total duration of traces in the two 3G datasets is more than 2000 minutes and the throughputs range between 0 and 10 Mbps. We split the original traces and generated more than 1000 traces with a duration of 240 seconds. We used a 4:1 ratio for training and testing.

Video \ Overhead	$v(1)$	$v(2)$
BigBuckBunny	18.8%	33.5%
ElephantDreams	18.5%	35.9%
BlueSky	21.9%	38.9%
DucksTakeOff	10.9%	20.6%

Table 1: *Coding overhead of three-layer SVC.* *We investigated the ranges of overhead $v(1)$, $v(2)$ and use them as the setup in the simulation testbed to learn an ABR algorithm for hybrid coding.*

ABR algorithms. We evaluated a number of ABR algorithms designed for AVC-encoded and hybrid-encoded videos. Note that we do not consider algorithms designed for vanilla SVC in the emulation as simulation results (Section 4.2) showed that the coding overhead made them perform much worse than our algorithm, which can also be inferred from Figure 5 and Table 1. Specifically, we compared our algorithm Grad-HYBJ to four state-of-the-art *AVC-based algorithms* including *(i)* Pensieve [18]: a DRL-based algorithm that includes only the limited decision space; *(ii)* MPC [35]: a control-theoretical approach relying on the prediction of bandwidth; *(iii)* BOLA [32]: a buffer-based algorithm that makes bitrate decisions solely basing on the state of the buffer; and *(iv)* BFAST [31]: an extension of BOLA that leverages heuristic to upgrade qualities of buffered segments. We used the robust version of MPC and the version of BFAST in *dash.js* Version 2.4 [1].

We extended BFAST to work with our HYBJ, i.e., achieving quality upgrade by downloading enhancement layers instead of AVC segments. We also included another algorithm LAAVS [22] that was designed for another hybrid coding, HYBP, for comparison. Briefly, this hybrid coding works by generating base layers for all supported qualities using AVC, and several enhancement layers to progressively upgrade the quality of each base layer level-by-level. For example, to support four quality levels, HYBP will generate $\{BL_m, 1 \le m \le 4\}$ and $\{EL_i^{(m,(m+i-1,m+i))}, 1 \le i \le 2\}$.

To use those algorithms for evaluation, we retrained Pensieve and optimized the hyperparameters of other algorithms for our QoE function. We denote each ABR algorithm and its targeted coding as *ABR-coding*, e.g., Pensieve-AVC vs. our proposed Grad-HYBJ.

Performance metrics. User QoE of a video session can be quantified based on the quality of each segment played back. We reported both the overall QoE score (as described in Equation (1)) and its three components including *bitrate utility*, *rebuffering penalty*, *smoothness penalty*. Further, we looked at five commonly used metrics[16, 20, 24, 25, 30]: *(i) average bitrate* and *(ii) standard deviation bitrate* of the segments in a video session; *(iii)* total *rebuffering time* in a video session; *(iv)* quality *switch times* in a video session; and *(v) switch amplitude* describing the bitrate improvement of a

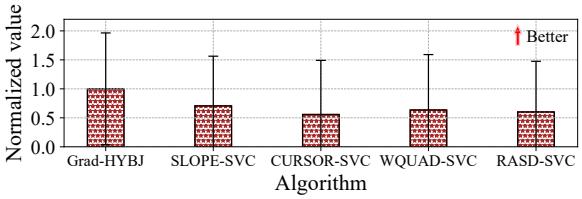


Figure 7: *Comparison with algorithms design for vanilla SVC in simulation.* *Even with an optimistic overhead assumption of* 10%*, the best performance of the algorithms can only reach 70.5% of that of our Grad-HYBJ .*

switch. Every test on a single trace is a video session, and the results are the average value of all the video sessions.

We also evaluated *bandwidth* and *storage* cost for each ABR algorithm with its respective video coding. *Bandwidth cost* is measured as the average total bits downloaded in a video session, while *storage cost* is the total storage size for storing all segments belong to one video. Both metrics are of interests to video streaming providers.

Additionally, we quantified the delay between when the bandwidth increases and when a user experiences higher quality segments. This is an important metric for its impact on user perceived quality improvement as pointed out in prior work [31]. We evaluated this delay separately as it is difficult to be incorporated into the overall QoE formula. We measured such delays under two conditions: *(i)* when a user experienced the *highest* quality that can be supported by the increased bandwidth, denoted as *reaction time*; *(ii)* when a user experienced *any* higher quality segments after the bandwidth increases, denoted as *reaction to higher time*.

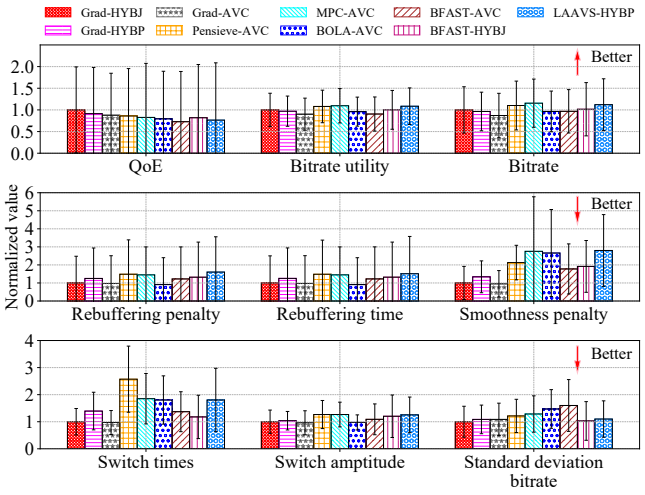
4.2 Comparison with Algorithms Designed for Vanilla SVC

We compared Grad-HYBJ with four algorithms designed for vanilla SVC (SLOPE [5], CURSOR [8], WQUAD [16], and RASD [25]) in the simulation testbed. The overhead ratio ω was configured to be 10%, which is an optimistic assumption according to [17] and Table 1. Results in Figure 7 demonstrate that even with an overhead ratio of 10%, the highest QoE obtained by those algorithms can only reach 70.5% of that obtained by our Grad-HYBJ. It indicates that these algorithms would not match up to our approach in the emulation. The same conclusion can be inferred from the results in Figure 5 and Table 1. In fact, many software encoders only support a limited number of enhancement layers due to the large overhead, e.g., the open source soft encoder *JSVM* allows two enhancement layers at maximum. This further poses a technical limitation for implementing those algorithms in the emulation.

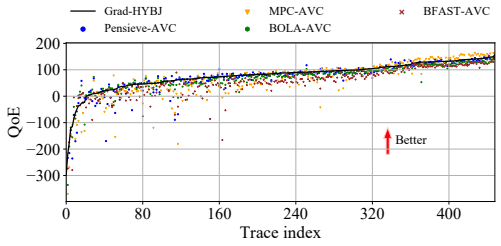
4.3 Quantifying QoE and its Breakdown

Figure 8(a) compares the overall QoE and its breakdown for existing ABR algorithm and coding method combinations, normalized to our Grad-HYBJ. We make the following three key observations.

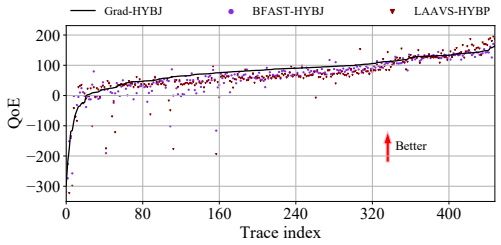
First, our Grad in conjunction with HYBJ achieved the best overall QoE score compared to state-of-the-art AVC-based DASH methods. Specifically, Grad-HYBJ outperformed Pensieve-AVC, the best



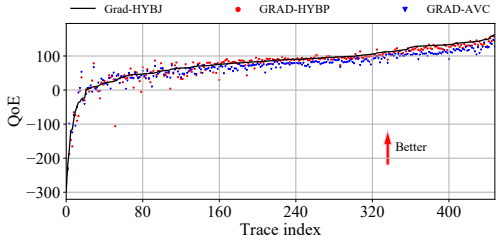
(a) QoE-related metrics.



(b) QoE distribution (AVC-based algorithms).



(c) QoE distribution (other hybrid alternatives).



(d) QoE distribution (Grad with other coding methods).

Figure 8: *QoE comparison of different DASH methods.* *Note that the large error bars in 8(a) represent the QoE variations under different network conditions instead of performance fluctuations. 8(b)-8(d) further compares the algorithms in terms of their performance on the same trace. The traces are indexed basing on the QoE that our Grad-HYBJ obtained on them.*

performing state-of-the-art method, by 17%. Note that the large error bars in Figure 8(a) represent the QoE variations under different network traces rather than performance fluctuations inherent to the algorithms. Figure 8(b) to Figure 8(d) further compares the algorithms in terms of their performance on the same trace, and the results show that Grad-HYBJ outperformed others most of the time. The performance gains of Grad-HYBJ can be largely attributed to the rebuffering time reduction and improved video smoothness, achieving the lowest *switch times* and *standard deviation bitrate*. However, as a combined effect of overhead incurred by quality upgrade of buffered segments and intrinsic tradeoff policy of Grad-HYBJ, Grad-HYBJ had 7.3% and 8.7% lower average *bitrate utility* compared to Pensieve-AVC and MPC-AVC. Additionally, Grad-HYBJ achieved comparable *switch amplitude* that is only higher than that of BOLA-AVC by 2.6%, due to its ability to adjust bitrate decisions after obtaining information of the relatively long-term bandwidth and thus distribute bandwidth more evenly to each segment.

Second, we observed that Grad-HYBJ achieved 13.8% and 9.4% higher QoE than Grad-AVC and Grad-HYBP, respectively. In particular, due to the high overhead for quality upgrade, Grad-AVC performed poorly for *bitrate utility*, which is 9.9% lower compared to Grad-HYBJ. Note that Grad-HYBJ performed better in all QoE-related metrics than Grad-HYBP, as HYBJ is more flexible for quality grade and can reduce associated coding overhead more effectively.

Third, Grad-HYBJ outperformed both BFAST-HYBJ and LAAVS-HYBP, whose ABR algorithms also include quality upgrade mechanism, by 22.1% and 30.8%. The breakdown shows that neither BFAST-HYBJ nor LAAVS-HYBP were able to balance across important QoE-related metrics. Further, using Grad-HYBP resulted in 19.4% higher QoE compared with LAAVS-HYBP, highlighting the effectiveness of our RL-based ABR design.

In the next two sections, we evaluated the QoE performance of our algorithm with smaller buffers (Section 4.4) and with distributional shift between the training and test sets (Section 4.5). We observe similar gains can be obtained with small buffers and our algorithm performs better than Pensieve-AVC when trained on network traces with low variance and tested on traces with high variance.

4.4 Performance with Smaller Buffers

Even though large buffers are more resistant to bandwidth fluctuations, smaller buffers are preferred sometimes as they can reduce the waste of bandwidth when users decide to quit watching early. In this part we study the performance of our Grad-HYBJ with more limited buffer sizes, in comparison with state-of-the-art ABR algorithms. We reconfigured the size of the buffer to be 32*s* and 16*s* in the emulation. The results are presented in Figure 9 (note that we omit the large error bars in the figure for a clearer comparison), and we can see that our Grad-HYBJ still outperforms the best performing state-of-the-art algorithm by 19.3% and 17.5% respectively in the two settings. The results suggest that similar gains still exist with our Grad-HYBJ when smaller buffers are deployed.

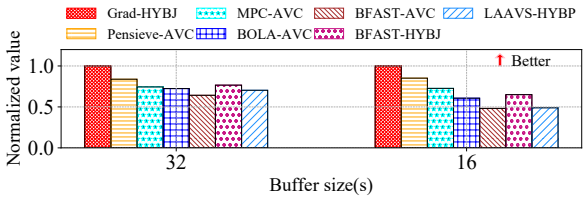


Figure 9: *QoE performance with smaller buffers.* *Similar gains can still be obtained by our Grad-HYBJ when smaller buffers are deployed. The large error bars are omitted for a clearer comparison.*

4.5 Implication of Distribution Shift

One challenge in developing learning-based ABR algorithms is the requirement for a dataset that can represent the real network conditions well. Such dataset is usually hard to obtain and the performance of the algorithms trained on less desirable dataset can degrade once deployed. Therefore, the algorithm's ability to generalize across different or even unfamiliar network scenarios is deemed important. We evaluated the ability of our algorithm to generalize compared with Pensieve-AVC, by training and testing on two datasets whose distributions differ from each other. To generate the two datasets, we sorted all our network traces basing on their bandwidth standard deviation and split them into two sets that are equal in size, and then we further divided each set into training set and test set. We denote the two datasets as **st** and **unst**, and they represent network conditions of relatively low and high variance, respectively. The detailed features of the two datasets are as follows:

- **st**: average bandwidth 4.4 Mbps, bandwidth standard deviation 1.7.
- **unst**: average bandwidth 5.5 Mbps, bandwidth standard deviation 5.6.

From the results in Figure 10 we observe that: *(i)* the distribution shift between the training and testing sets causes a noticeable degradation in performance for both algorithms, and the degradation is more severe when the algorithms are trained on **st** and tested on **unst**. *(ii)* When trained on **st** and tested on **unst**, our Grad-HYBJ suffers from a smaller performance degradation compared with Pensieve-AVC. The results suggest that our Grad-HYBJ performs better than Pensieve-AVC when trained on network conditions of low variance and generalize to those of high variance.

4.6 Impacts on Bandwidth and Storage Costs

Bandwidth cost. Figure 11 compares the bandwidth cost of each algorithm for video downloading. *First,* Grad-HYBJ had comparable bandwidth cost compared to the best performing state-of-the-art method Pensieve-AVC, achieving 17.0% increase in QoE with only 2.2% more transmitted data. Compared to the most bandwidth-efficient method BOLA-AVC, Grad-HYBJ consumed 12.7% more data but achieved 25.9% higher QoE. Grad-AVC executes quality upgrade but has negligible improvement in QoE over other AVC-based algorithms. The overhead of its quality upgrade neutralizes most of the benefits in QoE and lead to its extremely high bandwidth cost. *Second,* Grad-HYBJ had a bandwidth cost comparable to or even lower than other hybrid coding based alternatives, i.e.,1.7% and 4.9%

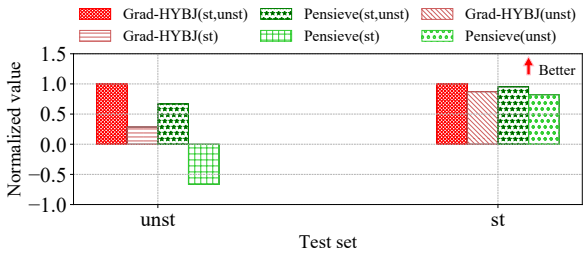


Figure 10: ***QoE performance under distribution shift between training set and test set.*** *The training set of each algorithm is given in the bracket. Our Grad-HYBJ performs better than Pensieve-AVC when trained on network conditions of low variance and generalize to those of high variance. The large error bars are omitted for a clearer comparison.*

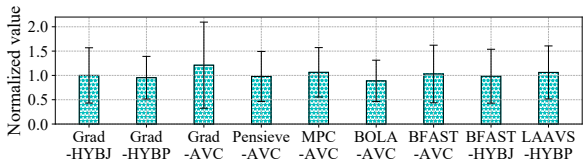


Figure 11: ***Bandwidth cost.***

higher than Grad-HYBP and BFAST-HYBJ, 6.2% lower than LAAVS-HYBP, with its achieved QoE being 9.5%, 22.1% and 30.8% higher repectively. In summary, our method Grad-HYBJ only incurred modest increase in bandwidth cost, in exchange for substantial QoE improvement.

Storage cost. Through our evaluation, we found that it takes 8X more storage space to store video segments generated by our HYBJ than by AVC, for six quality levels. However, we observed that *95.7%* of all the quality upgrades were made through one of the following seven base and enhancement layer combinations: $\{(BL_4, EL_1^{(4,(4,6))}), (BL_3, EL_1^{(3,(3,6))}), (BL_5, EL_1^{(5,(5,6))}),$ $(BL_3, EL_1^{(3,(3,5))}), (BL_3, EL_1^{(3,(3,4))}, EL_2^{(3,(4,5))}), (BL_1, EL_1^{(1,(1,2))}),$ $(BL_2, EL_2^{(2,(2,3))})\}$. Our observation suggests the potential to reduce storage cost by only keeping enhancement layers in the most frequently used combinations and all base layers. For example, with only the enhancement layers in the above seven combinations, we can reduce the storage cost of Grad-HYBJ to 2.27X of that required by AVC and still achieve 13.9% higher QoE than the best performing state-of-the-art method. As the unit storage cost is likely to halve every two years, based on Moore's Law, we think trading off storage for improved QoE can be reasonable for video streaming providers.

4.7 Comparisons of Reaction Time

Next, we study the reaction time achieved by different combinations of ABR algorithms and coding methods. We constructed 500 network traces, each with 300 seconds, from the FCC dataset [2], which contains traces of small durations with different but sustainable bandwidth. All constructed traces have an initial bandwidth of less than 0.6 Mbps and then an upward jump between 20 and 130 seconds. The increased bandwidth is sustainable and higher than

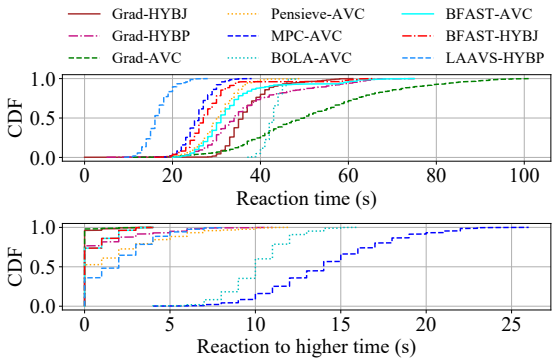


Figure 12: ***Reaction time evaluation.***

the highest bitrate at 4.3 Mbps, with a range of [7, 10] Mbps. Figure 12 shows the CDFs for two reaction time metrics. We observed that Grad-HYBJ and Grad-AVC had the fastest *reaction to higher time* for all tested traces. In most cases, Grad-HYBJ was able to upgrade video quality *immediately* after observing the bandwidth increase. Further, Grad-HYBJ achieved comparable *reaction time* to BFAST-AVC, with 3 seconds and 2 seconds differences in median and 95th percentile, respectively. Although LAAVS-HYBP had the fastest *reaction time*, it did so by aggressively upgrading without meticulously tracking the bandwidth fluctuations, as demonstrated by its lower QoE in Section 4.3. Combined, it demonstrates that our Grad-HYBJ was able to react to the bandwidth changes more discreetly and this allows it to better cope with different types of bandwidth increases, e.g., a sustainable improvement or a mere fluctuation, as suggested by its advantage in rebuffering time reduction and video smoothness improvement in Section 4.3.

5 CONCLUSION

In this work, we explored the design of ABR algorithms with quality upgrade mechanism for streaming SVC-encoded videos through a Deep Reinforcement Learning based approach Grad. We addressed two key challenges, namely enlarged decision space in ABR algorithm designing and coding overhead of SVC, through tailored action designs and hybrid coding. We evaluated the performance of our proposed Grad-HYBJ on an emulated streaming system, with commonly used network traces and video. Grad-HYBJ outperformed state-of-the-art AVC-based methods including Pensieve-AVC by at least 17.0% in average QoE. Additionally, Grad-HYBJ required 12.7% more transmitted data, but achieved 25.9% higher QoE compared to the most bandwidth-efficient method BOLA with AVC. Lastly, Grad-HYBJ reacted to bandwidth increase more discreetly and its storage cost can be reduced to 2.27X of that of AVC-based methods while still maintaining a 13.9% improvement in QoE. To sum up, our work demonstrated an effective way to design ABR algorithms with quality upgrade mechanism and to utilize SVC to benefit video streaming in practice.

REFERENCES

[1] 2016. Dash.js. https://github.com/Dash-Industry-Forum/dash.js/
[2] 2017. Federal Communications Commission. 2017. Raw Data - Measuring Broadband America. https://www.fcc.gov/reports-research/reports

[3] 2019. Cisco Visual Networking Index: Forecast and Trends, 2017–2022 White Paper. https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.html

[4] Zahaib Akhtar, Yun Seong Nam, Ramesh Govindan, Sanjay Rao, Jessica Chen, Ethan Katz-Bassett, Bruno Ribeiro, Jibin Zhan, and Hui Zhang. 2018. Oboe: Auto-Tuning Video ABR Algorithms to Network Conditions. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. 44–58.

[5] Travis Andelin, Vasu Chetty, Devon Harbaugh, Sean Warnick, and Daniel Zappala. 2012. Quality Selection for Dynamic Adaptive Streaming over HTTP with Scalable Video Coding. In *Proceedings of the 3rd Multimedia Systems Conference*. 149–154.

[6] G. Bjontegaard. 2001. Calculation of average PSNR differences between RD-Curves. *Proceedings of the ITU-T Video Coding Experts Group (VCEG) Thirteenth Meeting* (01 2001).

[7] Ayub Bokani, Mahbub Hassan, Salil S. Kanhere, Jun Yao, and Garson Zhong. 2016. Comprehensive Mobile Bandwidth Traces from Vehicular Networks. In *Proceedings of the 7th International Conference on Multimedia Systems*. Article 44, 6 pages.

[8] Niels Bouten, Steven Latré, Jeroen Famaey, Filip De Turck, and Werner Van Leekwijck. 2013. Minimizing the Impact of Delay on Live SVC-based HTTP Adaptive Streaming Services. *Proceedings of the 2013 IFIP/IEEE International Symposium on Integrated Network Management, IM 2013*.

[9] Federico Chiariotti, Stefano D'Aronco, Laura Toni, and Pascal Frossard. 2016. Online Learning Adaptation Strategy for DASH Clients. In *Proceedings of the 7th International Conference on Multimedia Systems*. Article 8, 12 pages.

[10] Maxim Claeys, Steven Latré, Jeroen Famaey, Tingyao Wu, Werner Van Leekwijck, and Filip De Turck. 2013. Design of a Q-Learning based client quality selection algorithm for HTTP Adaptive Video Streaming.

[11] Maxim Claeys, Steven Latré, Jeroen Famaey, TY Wu, W Van Leekwijck, and Filip De Turck. 2014. Design and optimisation of a (FA)Q-learning-based HTTP adaptive streaming client. *CONNECTION SCIENCE* 26, 1 (2014), 22.

[12] Florin Dobrian, Vyas Sekar, Asad Awan, Ion Stoica, Dilip Antony Joseph, Aditya Ganjam, Jibin Zhan, and Zhang Hui. 2011. Understanding the impact of video quality on user engagement. In *Acm Sigcomm Conference*. 362–373.

[13] A. Elgabli, V. Aggarwal, S. Hao, F. Qian, and S. Sen. 2018. LBP: Robust Rate Adaptation Algorithm for SVC Video Streaming. *IEEE/ACM Transactions on Networking* 26, 4 (2018), 1633–1645.

[14] Michael Grafl, Christian Timmerer, Hermann Hellwagner, Wael Chérif, and Adlen Ksentini. 2013. Evaluation of Hybrid Scalable Video Coding for HTTP-based Adaptive Media Streaming with High-Definition Content. *2013 IEEE 14th International Symposium on a World of Wireless, Mobile and Multimedia Networks, WoWMoM 2013*, 7.

[15] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. 2014. A Buffer-Based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service. In *Proceedings of the 2014 ACM Conference on SIGCOMM*. 187–198.

[16] Jaehyun Hwang, Junghwan Lee, and Chuck Yoo. 2016. Eliminating bandwidth estimation from adaptive video streaming in wireless networks. *Signal Processing: Image Communication* 47 (2016), 242 – 251.

[17] Christian Kreuzberger, Daniel Posch, and Hermann Hellwagner. 2015. A Scalable Video Coding Dataset and Toolchain for Dynamic Adaptive Streaming over HTTP. In *Proceedings of the 6th ACM Multimedia Systems Conference*. 213–218.

[18] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. 2017. Neural Adaptive Video Streaming with Pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. 197–210.

[19] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous Methods for Deep Reinforcement Learning. In *Proceedings of The 33rd International Conference on Machine Learning*.

[20] Ricky K. P. Mok, Xiapu Luo, Edmond W. W. Chan, and Rocky K. C. Chang. 2012. QDASH: A QoE-Aware DASH System. In *Proceedings of the 3rd Multimedia Systems Conference*. 11–22.

[21] Afshin Taghavi Nasrabadi, Anahita Mahzari, Joseph D. Beshay, and Ravi Prakash. 2017. Adaptive 360-Degree Video Streaming Using Scalable Video Coding. In *Proceedings of the 25th ACM International Conference on Multimedia*. 1689–1697.

[22] Afshin Taghavi Nasrabadi and Ravi Prakash. 2018. Layer-Assisted Adaptive Video Streaming. In *Proceedings of the 28th ACM SIGMM Workshop on Network and Operating Systems Support for Digital Audio and Video*. 31–36.

[23] Ravi Netravali, Anirudh Sivaraman, Keith Winstein, Somak Das, Ameesh Goyal, and Hari Balakrishnan. 2014. Mahimahi: A Lightweight Toolkit for Reproducible Web Measurement. In *Proceedings of the 2014 ACM Conference on SIGCOMM*. 129–130.

[24] Pengpeng Ni, Ragnhild Eg, Alexander Eichhorn, Carsten Griwodz, and Pål Halvorsen. 2011. Flicker Effects in Adaptive Video Streaming to Handheld Devices. In *Proceedings of the 19th ACM International Conference on Multimedia*. 463–472.

[25] S. G. Ozcan, T. Kivilcim, C. Cetinkaya, and M. Sayit. 2017. Rate adaptation algorithm with backward quality increasing property for SVC-DASH. In *2017 IEEE 7th International Conference on Consumer Electronics - Berlin (ICCE-Berlin)*. 24–28.

[26] Darijo Raca, Jason J. Quinlan, Ahmed H. Zahran, and Cormac J. Sreenan. 2018. Beyond Throughput: A 4G LTE Dataset with Channel and Context Metrics. In *Proceedings of the 9th ACM Multimedia Systems Conference*. 460–465.

[27] Haakon Riiser, Paul Vigmostad, Carsten Griwodz, and Pål Halvorsen. 2013. Commute Path Bandwidth Traces from 3G Networks: Analysis and Applications. In *Proceedings of the 4th ACM Multimedia Systems Conference*. 114–118.

[28] S. Rimac-Drlje, O. Nemcic, and M. Vranjes. 2008. Scalable Video Coding extension of the H.264/AVC standard. In *Elmar, International Symposium*. 9 – 12.

[29] Heiko Schwarz, Detlev Marpe, and Thomas Wiegand. 2007. Overview of the Scalable Video Coding Extension of the H.264/AVC Standard. *Circuits and Systems for Video Technology, IEEE Transactions on* 17 (10 2007), 1103 – 1120.

[30] C. Sieber, T. Hoßfeld, T. Zinner, P. Tran-Gia, and C. Timmerer. 2013. Implementation and user-centric comparison of a novel adaptation logic for DASH with SVC. In *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*. 1318–1323.

[31] Kevin Spiteri, Ramesh Sitaraman, and Daniel Sparacio. 2018. From theory to practice: improving bitrate adaptation in the DASH reference player. In *Proceedings of the 9th ACM Multimedia Systems Conference*. 123–137.

[32] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman. 2016. BOLA: Near-optimal bitrate adaptation for online videos. In *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*. 1–9.

[33] Gary Sullivan, Pankaj Topiwala, and Ajay Luthra. 2004. The H.264/AVC Advanced Video Coding Standard: Overview and Introduction to the Fidelity Range Extensions. *Proceedings of SPIE - The International Society for Optical Engineering* (11 2004).

[34] Siyuan Xiang, Min Xing, Lin Cai, and Jianping Pan. 2015. Dynamic Rate Adaptation for Adaptive Video Streaming in Wireless Networks. *Signal Processing: Image Communication* 39 (09 2015).

[35] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. 2015. A Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP. In *Acm Conference on Special Interest Group on Data Communication*. 325–338.