Comyco: Quality-Aware Adaptive Video Streaming via Imitation Learning

Tianchi Huang[1], Chao Zhou[2*], Rui-Xiao Zhang[1], Chenglei Wu[1], Xin Yao[1], Lifeng Sun[3*]

[1]Dept. of Computer Science and Technology, Tsinghua University
[2]Beijing Kuaishou Technology Co., Ltd., China
[3]BNRist, Dept. of Computer Science and Technology, Tsinghua University
{htc17@mails.,sunlf@}tsinghua.edu.cn, zhouchao@kuaishou.com

ABSTRACT

Learning-based Adaptive Bit Rate (ABR) method, aiming to learn outstanding strategies without any presumptions, has become one of the research hotspots for adaptive streaming. However, it is still suffering from several issues, i.e., low sample efficiency and lack of awareness of the video quality information. In this paper, we propose *Comyco*, a video quality-aware ABR approach that enormously improves the learning-based methods by tackling the above issues. Comyco trains the policy via imitating expert trajectories given by the instant solver, which can not only avoid redundant exploration but also make better use of the collected samples. Meanwhile, Comyco attempts to pick the chunk with higher perceptual video qualities rather than video bitrates. To achieve this, we construct Comyco's neural network architecture, video datasets and QoE metrics with video quality features. Using trace-driven and real world experiments, we demonstrate significant improvements of Comyco's sample efficiency in comparison to prior work, with 1700x improvements in terms of the number of samples required and 16x improvements on training time required. Moreover, results illustrate that Comyco outperforms previously proposed methods, with the improvements on average QoE of 7.5% - 16.79%. Especially, Comyco also surpasses state-of-the-art approach Pensieve by 7.37% on average video quality under the same rebuffering time.

CCS CONCEPTS

• **Information systems** → *Multimedia streaming*; • **Computing methodologies** → *Neural networks*;

KEYWORDS

Imitation Learning, Quality-aware, Adaptive Video Streaming

---

* Corresponding Author.

1 INTRODUCTION

Recent years have seen a tremendous increase in the requirements of watching online videos [11]. Adaptive bitrate (ABR) streaming, the method that dynamically switches download chunk bitrates for restraining rebuffering event as well as obtaining higher video qualities, has become the popular scheme to deliver videos with high quality of experience (QoE) to the users [10]. Recent model-based ABR approaches (§7) pick the next chunk's video bitrate via only current network status [23], or buffer occupancy [44], or joint consideration of both two factors[51]. However, such heuristic methods are usually set up with presumptions, that fail to work well under unexpected network conditions [28]. Thus, learning-based ABR methods adopt reinforcement learning (RL) method to *learn* the strategies without any presumptions, which outperform traditional model-based approaches.

Nevertheless, learning-based ABR methods suffer from two key issues. While recent work [15, 28] often adopts RL methods to train the neural network, such methods lack the efficiency of both collected and exploited expert samples, which leads to the inefficient training [30]. Besides, the majority of existing ABR approaches [6, 28, 51] neglect the video quality information, while perceptual video quality is a non-trivial feature for evaluating QoE (§5.1,[19]). Thus, despite their abilities to achieve higher QoE objectives, such schemes may generate a strategy that diverges from the actual demand (§2.2).

In this paper, we propose *Comyco*, a novel video quality-aware learning-based ABR system, aiming to remarkably improve the overall performance of ABR algorithms via tackling the above challenges. Unlike previous RL-based schemes [28], Comyco leverages imitation learning [35] to train the neural network (NN). That is because the near-optimal policy can be precisely and instantly estimated via the current state in the ABR scenario and the collected expert policies can enable the NN for fast learning. Following this thought (§3.1), the agent is allowed to *explore* the environment and *learn* the policy via the expert policies given by the solver (§4.5). Specifically, we propose *instant solver* (§4.2) to estimate the expert action with a faithful *virtual player* (§6.1). Furthermore, we utilize *experience replay buffer* (§4.4) to store expert policies and train the NN via the specific loss function L_{comyco} (§4.3).

Besides, Comyco aims to select bitrate with high perceptual video quality rather than high video bitrate. To achieve this, we first integrate the information of video contents, network status, and video playback states into the Comyco's NN for bitrate selection (§4.1). Next, we consider using VMAF [38], an objective full-reference perceptual video quality metric, to measure the video quality. Concurrently, we also propose a linear combination of video quality-based

QoE metric that achieves the state-of-art performance on Waterloo Streaming SQoE-III [12] dataset (§5.1). Finally, we collect a DASH-video dataset with various types of videos, including movies, sports, TV-shows, games, news, and music videos (MV) (§5.2).

Using trace-driven emulation (§6.1), we find that Comyco significantly accelerates the training process, with 1700x improvements in terms of number of samples required compared to recent work (§6.2). Comparing Comyco with existing schemes under various network conditions (§6.1) and videos (§5.2), we show that Comyco outperforms previously proposed methods, with the improvements on average QoE of 7.5% - 16.79%. In particular, Comyco performs better than state-of-the-art learning-based approach Pensieve, with the improvements on the average video quality of 7.37% under the same rebuffering time. Further, we present results which highlight Comyco's performance with different hyperparameters and settings (§6.4). Finally, we validate Comyco in real world network scenarios (§6.5). Extensive results indicate the superiority of Comyco over existing state-of-the-art approaches.

In general, we summarize the contributions as follows:

▷ We propose Comyco, a video quality-aware learning-based ABR system, that significantly ameliorates the weakness of the learning-based ABR schemes from two perspectives.
▷ To the best of our knowledge, we are the first to leverage imitation learning to accelerate the training process for ABR tasks. Results indicate that utilizing imitation learning can not only achieve fast convergence rates but also improve performance.
▷ Unlike prior work, Comyco picks the video chunk with high perceptual video quality instead of high video bitrate. Results also demonstrate the superiority of the proposed algorithm.

2 BACKGROUND AND CHALLENGES

2.1 ABR Overview

Due to the rapid development of network services, watching video online has become a common trend. Today, the predominant form for video delivery is adaptive video streaming, such as HLS (HTTP Live Streaming) [2] and DASH [1], which is a method that dynamically selects video bitrates according to network conditions and clients' buffer occupancy. Traditional video streaming framework consists of a video player client with a constrained buffer length and an HTTP-Server or Content Delivery Network (CDN). The video player client decodes and renders video frames from the playback buffer. Once the streaming service starts, the client fetches the video chunk from the HTTP Server or CDN in order by an ABR algorithm. Meanwhile, the algorithm, deployed on the client side, determines the next chunk N and next chunk video quality Q_N via throughput estimation and current buffer utilization. The goal of the ABR algorithm is to provide the video chunk with high qualities and avoid stalling or rebuffering [10].

2.2 Challenges for learning-based ABRs

Most traditional ABR algorithms [23, 44, 51] leverage time-series prediction or automation control method to make decisions for the next chunk. Nevertheless, such methods are built in pre-assumptions that it is hard to keep its performance in all considered network scenarios [28]. To this end, learning-based ABR algorithms [15, 18, 28] are proposed to solve the problem from another perspective: it
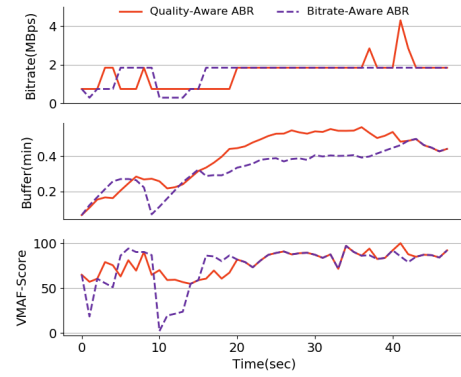

Figure 1: We evaluate quality-aware ABR algorithm and bitrate-aware ABR algorithm with the same video on Norway network traces respectively. Results are plotted as the curves of selected bitrate, buffer occupancy and the selected chunk's VMAF (§5.1,[38]) for entire sessions.

adopts deep reinforcement learning (DRL) to train a neural network (NN) from scratch towards the better QoE objective. Despite the outstanding results that recent work has obtained, learning-based ABR methods suffer from several key issues:

The weaknesses of RL-based ABR algorithms. Recent learning-based ABR schemes often adopt RL methods to maximize the average QoE objectives. During the training, the agent rollouts a trajectory and updates the NN with policy gradients. However, the effect of calculated gradients heavily depends on the amount and quality of collected experiences. In most cases, the collected samples seldom stand for the optimal policy of the corresponding states, which leads to a long time to converge to the sub-optimal policy [29, 35]. Thus, we are facing the first challenge: *Considering the characteristic of ABR tasks, can we precisely estimate the optimal direction of gradients to guide the model for better updating?*

The unique video quality. What's more, previous learning-based ABR schemes [28, 51] are evaluated by typical QoE objectives that use the combination of video bitrates, rebuffering times and video smoothness. However, such QoE metrics are short-handed because these forms of parameters neglect the quality of video presentations [49]. Meanwhile, recent work [13, 37] has found that perceptual video quality features play a vital part in evaluating the performance of VBR-encoded ABR streaming services. To prove this, we plot the trajectory generated by the quality-aware ABR and bitrate-aware algorithm on Figure 1. As shown, the bitrate-aware algorithm selects the video chunk with higher bitrate but neglects the corresponding video quality, resulting in a large fluctuation in the perceptual video qualities. What's more, bitrate-aware algorithm often wastes the buffer on achieving a slight increase in video quality, which may cause unnecessary stalling event. On the contrast, the quality-aware algorithm picks the chunk with high and stable perceptual video quality and preserves the buffer occupancy within an allowable range. To this end, one of the better solutions is to add video bitrates as another metric to describe the perceptual video quality. We, therefore, encounter the second challenge of our work: *How to construct a video quality-aware ABR system?*

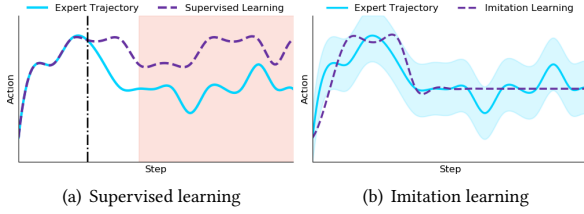(a) Supervised learning (b) Imitation learning

Figure 2: The real trajectory on the ABR task given by imitation learning and supervised learning, where the red background means the player occurs the rebuffering event.

3 METHODS

Motivated by the key challenges (§2.2), we propose Comyco, a video quality-aware learning-based ABR scheme. In this section, we introduce two main ideas of Comyco: training NN via imitation learning (§3.1) and a complete video quality-based ABR system (§3.2).

3.1 Training ABRs via Imitation Learning

Recall that the key principle of RL-based method is to maximize *reward* of each *action* taken by the agent in given *states* per step, since the agent doesn't really know the optimal strategy [45]. However, recent work [6, 18, 28, 36, 43, 51] has demonstrated that the ABR process can be precisely emulated by an offline virtual player (§6.1) with complete future network information. What's more, by taking several steps ahead, we can further accurately *estimate* the near-optimal expert policy of any ABR state within an acceptable time (§4.2). To this end, the intuitive idea is to leverage supervised learning methods to minimize the loss between the predicted and the expert policy. Nevertheless, it's impractical because the off-policy method [45] suffers from *compounding error* when the algorithm executes its policy, leading it to drift to new and unexpected states [25]. For example, as shown in Figure 2[a], in the beginning, supervised learning-based ABR algorithm fetches the bitrate that is consistent with the expert policy, but when it selects a bitrate with a minor error (after the black line), the state may be transitted to the situation not included in the dataset, so the algorithm would select another wrong bitrate. Such compounding errors eventually lead to a continuous rebuffering event. As a result, supervised-learning methods cannot learn to recover from failures.

In this paper, we aim to leverage imitation learning, a method that closely related to RL and supervised learning, to learn the strategy from the expert policy samples. Imitation learning method reproduces desired behavior according to expert demonstrations [35]. The key idea of imitation learning is to allow the NN to explore environments and collect samples (just like RL) and learn the policy based on the expert policy (just as supervised learning). In detail, at step t, the algorithm infers a policy π_t at ABR state S_t. It then computes a loss $l_t(\pi_t, \pi_t^*)$ w.r.t the expert policy π_t^*. After observing the next state S_{t+1}, the algorithm further provides a different policy π_{t+1} for the next step $t+1$ that will incur another loss $l_t(\pi_{t+1}, \pi_{t+1}^*)$. Thus, for each π_t in the class of policies $T \in \{\pi_0, \dots, \pi_t\}$, we can find the policy $\hat{\pi}$ through any supervised learning algorithms (Eq. 1).
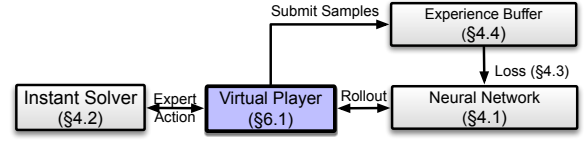

Figure 3: Comyco's Basic System Work-flow Overview. Training methodologies are available in §4.5.

$$\hat{\pi} = \underset{\pi \in T}{arg\,min}\, \mathbb{E}_{s \sim d_\pi}\left[l_t(\pi_t, \pi_t^*)\right] \tag{1}$$

Figure 2[b] elaborates the principle of imitation learning-based ABR schemes: the algorithm attempts to explore the strategy in a range near the expert trajectory to avoid compounding errors.

3.2 Video Quality-aware ABR System Setup

Our next challenge is to set up a video quality-aware ABR system. The work is generally composed of three tasks: 1) We construct Comyco's NN architecture with jointly considering several underlying metrics, i.e, past network features and video content features as well as video playback features (§4.1). 2) We propose a quality-based QoE metric (§5.1). 3) We collect a video quality DASH dataset which includes various types of videos (§5.2).

4 SYSTEM OVERVIEW

In this section, we describe the proposed system in detail. Comyco's basic system work-flow is illustrated in Figure 3. The system is mainly composed of a NN, an ABR virtual player, an instant solver, and an experience replay buffer. We start by introducing the Comyco's modules. Then we explain the basic training methodology. Finally, we further illustrate Comyco with a multi-agent framework.

4.1 NN Architecture Overview

Motivated by the recent success of on-policy RL-based methods, Comyco's learning agent is allowed to explore the environment via traditional rollout methods. For each epoch t, the agent aims to select next bitrate via a neural network (NN). We now explain the details of the agent's NN including its inputs, outputs, network architecture, and implementation.

Inputs. We categorize the NN into three parts, network features, video content features and video playback features ($S_k = \{C_k, M_k, F_k\}$). Details are described as follows.

▷ *Past Network features.* The agent takes past t chunks' network status vector $C_k = \{c_{k-t-1}, \dots, c_k\}$ into NN, where c_i represents the throughput measured for video chunk i. Specifically, c_i is computed by $c_i = n_{r,i}/d_i$, in which $n_{r,i}$ is the downloaded video size of chunk i with selected bitrates r, and d_i means download time for video chunk $n_{r,i}$.

▷ *Video content features.* Besides that, we also consider adding video content features into NN's inputs for improving its abilities on detecting the diversity of video contents. In details, the learning agent leverages $M_k = \{N_{k+1}, V_{k+1}\}$ to represent video content features. Here N_{k+1} is a vector that reflects the video size for each bitrate of the next chunk $k+1$, and V_{k+1} is a vector
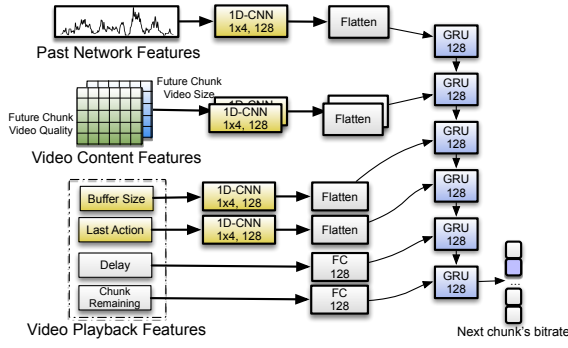
Figure 4: Comyco's NN architecture Overview.

which stands for the perceptual video quality metrics for each bitrate of the next chunk.

▷ *Video playback features.* The last essential feature for describing the ABR's state is the current video playback status. The status is represented as $F_k = \{v_{k-1}, B_k, D_k, m_k\}$, where v_{k-1} is the perceptual video quality metric for the past video chunk selected, B_k, D_k are vectors which stand for past t chunks' buffer occupancy and download time, and m_k means the normalized video chunk remaining.

Outputs. Same as previous work, we consider using discrete action space to describe the output. Note that the output is an n-dim vector indicating the probability of the bitrate being selected under the current ABR state S_k.

Implementation. As shown in Figure 4, for each input type, we use a proper and specific method to extract the underlying features. In details, we first leverage a single 1D-CNN layer with kernel=4, channels=128, stride=1 to extract network features to a 128-dim layer. We then use two 1D-CNN layers with kernel=1x4, channels=128 to fetch the hidden features from the future chunk's video content matrix. Meanwhile, we utilize 1D-CNN or fully connected layer to extract the useful characteristics from each metric upon the video playback inputs. The selected features are passed into a GRU layer and outputs as a 128-dims vector. Finally, the output of the NN is a *6-dims* vector, which represents the probabilities for each bitrate selected. We utilize *RelU* as the active function for each feature extraction layer and leverage *softmax* for the last layer.

4.2 Instant Solver

Once the sampling module rolls out an action a_t, we aim to design an algorithm to fetch all the *optimal* actions $\hat{a}_t$ with respect to current state s_t. Followed by these thoughts, we further propose *Instant Solver*. The key idea is to choose future chunk k's bitrate R_k by taking N steps ahead via an offline *virtual player*, and solves a specific *QoE maximization problem* with future network throughput measured C_t, in which the future real throughput can be successfully collected under both offline environments and real-world network scenarios. Inspired by recent model-based ABR work [51], we formulate the problem as demonstrated in Eq. 2, denoted as $QoE^N_{max}K$. In detail, the virtual player consists of a virtual time, a real-world network trace and a video description. At virtual time t_k, we first calculate download time for chunk k via $d_k(R_k)/C_k$, where d_k is the video chunk size for bitrate R_k, and C_k is average throughput

measured. We then update B_{k+1} buffer occupancy for chunk $k+1$, in which δt_k reflects the waiting time such as Round-Trip-Time (RTT) and video render time, and B_{max} is the max buffer size. Finally, we refresh the virtual time t_{k+1} for the next computation. Note that the problem can be solved with any optimization algorithms, such as memoization, dynamic programming as well as Hindsight [20]. Ideally, there exists a trade-off between the computation overhead and the performance. We list the performance comparison of instant solver with different N in §6.4. In this work, we set $N = 8$.

$$\max_{R_1, \ldots, R_k, T_s} QoE^N, \quad s.t.$$

$$\begin{cases} t_{k+1} = t_k + \dfrac{d_k(R_k)}{C_k} + \delta t_k, \\[2mm] C_k = \dfrac{1}{t_{k+1} - t_k - \delta t_k} \displaystyle\int_{t_k}^{t_{k+1} - \delta t_k} C_t\, dt, \\[2mm] B_{k+1} = \left[\left(B_k - \dfrac{d_k(R_k)}{C_k} \right)_+ + L - \delta t_k \right]_+, \\[2mm] B_1 = T_s, \\[1mm] B_k \in [0, B_{max}], R_k \in R, \forall k = 1, \ldots, N. \end{cases} \quad (2)$$

4.3 Choice of Loss Functions for Comyco

In this section, we start with designing the loss function from the fundamental RL training methodologies. The goal of the RL-based method is to maximize the Bellman Equation, which is equivalent to maximize the value function $q_\pi(s, a)$ [45]. The equation is listed in Eq. 3, where $q_*(s, a)$ stands for the maximum action value function on all policies, $V_\pi(s)$ is the value function, $\pi(s, a; \theta)$ is the rollout policy. Thus, given an expert action $q_\pi(s, \hat{a}) = q_*(s, a)$, we can update the model via minimizing the gap between the true action probability $\hat{A}$ and π, where A is an one hot encoding in terms of $\hat{a}$. In this paper, we use cross entropy error as the loss function. Recall that the function can be represented as any traditional behavioral cloning loss methods [35], such as Quadratic, LI-loss and Hinge loss function. In addition, we find that the other goal of the loss function is to maximize the probabilities of the selected action, while the goal significantly reduces the aggressiveness of exploration, and finally, resulting in obtaining the sub-optimal performance. Thus, motivated by the recent work on RL [31], we add the entropy of the policy π to the loss function. It can encourage the algorithm to increase the exploration rate in the early stage and discourage it in the later stage. The loss function for Comyco is described in Eq 4.

$$\max_\pi V_\pi(s) = \sum_{a \in A} \pi(a|s; \theta) q_\pi(s, a)$$
$$= \max_\pi \max_a q_\pi(s, a) \quad (3)$$
$$= \max_a q_*(s, a)$$

$$L_{comyco} = -\sum \hat{A} \log \pi(s, a; \theta) + \alpha H(\pi(s; \theta)). \quad (4)$$

Here $\pi(s, a; \theta)$ is the rollout policy selected by the NN, $\hat{A}$ is the real action probability vector generated by the expert actor $\hat{a}$, $H(\pi(s; \theta))$ represents the entropy of the policy, α is a hyperparameter which controls the encouragement of exploration. In this paper, we set $alpha = 0.001$ and discuss L_{comyco} with different α in §6.4.

4.4 Training Comyco with Experience Replay

Recent off-policy RL-based methods [32] leverage experience replay buffer to achieve better convergence behavior when training a
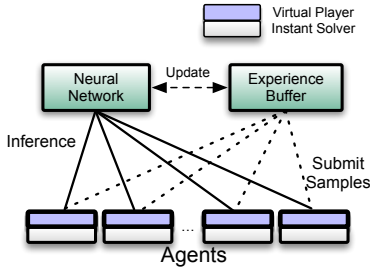
Figure 5: Comyco's Multi-Agent Framework Overview.

function approximator. Inspired by the success of these approaches, we also create a sample buffer which can store the past expert strategies and allow the algorithm to randomly picks the sample from the buffer during the training process. We will discuss the effect of utilizing experience replay on Comyco in §6.4.

4.5 Methodology

We summarize the Comyco's training methodology in Alg. 1.

Algorithm 1 Overall Training Procedure

Require: Training model π, Instant Solver(§4.2).
1: Sample Training Batch $B = \{\}$.
2: **procedure** TRAINING
3:    Initialize π.
4:    Get State ABR state s_t.
5:    **repeat**
6:        Picks a_t according to policy $\pi(s_t; \theta)$.
7:        Expert action $\hat{a}_t = $ Instant Solver(s_t).
8:        $B \leftarrow B \bigcup \{s_t, \hat{a}_t\}$.
9:        Samples a batch $\hat{B} \in B$.
10:        Updates network π with $\hat{B}$ using Eq.4;
11:        Produces next ABR state S_{t+1} according to s_t and a_t.
12:        $t \leftarrow t + 1$
13:    **until** Converged
14: **end procedure**

4.6 Parallel Training

It's notable that the training process can be designed asynchronously, which is quite suitable for multi-agent parallel training framework. Inspired by the multi-agent training method [19, 31], we modify Comyco's framework from single-agent training to multi-agent training. As illustrated in Figure 5, Comyco's multi-agent training consists of three parts, a central agent with a NN, an experience replay buffer, and a group of agents with a virtual player and an instant solver. For any ABR state s, the agents use virtual player to emulate the ABR process w.r.t current states and actions given by the NN which placed on the central agent, and collect the expert action $\hat{a}$ through the instant solver; they then submit the information containing $\{s, \hat{a}\}$ to the experience replay buffer. The central agent trains the NN by picking the sample batch from the buffer. Note that this can happen asynchronously among all agents. By default, Comyco uses **12** agents, which is the same number of CPU cores of our PC, to accelerate the training process.

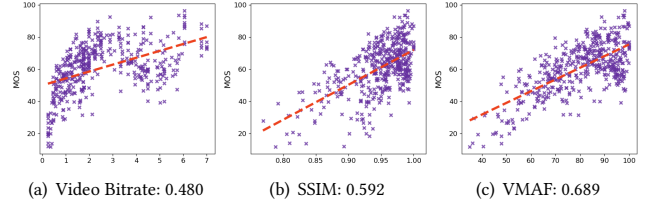(a) Video Bitrate: 0.480   (b) SSIM: 0.592   (c) VMAF: 0.689

Figure 6: Correlation comparison of video presentation quality metrics on the SQoE-III dataset [12]. Results are summarized by Pearson correlation coefficient [8].

4.7 Implementation

We now explain how to implement Comyco. We use TensorFlow [4] to implement the training workflow and utilizing TFlearn [47] to construct the NN architecture. Besides, we use C++ to implement instant solver and the virtual player. Then we leverage Swig [7] to compile them as a python class. Next, we will show more details: Comyco takes the past sequence length $k = 8$ (as suggested by [28]) and future 7 video chunk features (as suggested by [51]) into the NN. We set learning rate $\alpha = 10^{-4}$ and use Adam optimizer [24] to optimize the model. For more details, please refer to our repository [1].

5 QOE METRICS & VIDEO DATASETS

Upon constructing the Comyco's NN architecture with considering video content features, we have yet discussed how to train the NN. Indeed, we lack a video quality-aware QoE model and an ABR video dataset with video quality assessment. In this section, we use VMAF to describe the perceptual video quality of our work. We then propose a video quality-aware QoE metric under the guidance of real-world ABR QoE dataset [12]. Finally, we collect and publish a DASH video dataset with different VMAF assessments.

5.1 QoE Model Setup

Motivated by the linear-based QoE metric that widely used to evaluate several ABR schemes [6, 9, 28, 36, 37, 51], we concluded our QoE metric QoE_v as:

$$\text{QoE}_v = \alpha \sum_{n=1}^{N} q(R_n) - \beta \sum_{n=1}^{N} T_n$$
$$+ \gamma \sum_{n=1}^{N-1} [q(R_{n+1}) - q(R_n)]_+ - \delta \sum_{n=1}^{N-1} [q(R_{n+1}) - q(R_n)]_-, \quad (5)$$

where N is the total number of chunks during the session, R_n represents the each chunk's video bitrate, T_n reflects the rebuffering time for each chunk n, $q(R_n)$ is a function that maps the bitrate R_n to the video quality perceived by the user, $[q(R_{n+1}) - q(R_n)]_+$ denotes positive video bitrate smoothness, meaning switch the video chunk from low bitrate to high bitrate and $[q(R_{n+1}) - q(R_n)]_-$ is negative smoothness. Note that $\alpha, \beta, \gamma, \delta$ are the parameters to describe their aggressiveness.

Choice of $q(R_n)$. To better understand the correlation between video presentation quality and QoE metric, we test the correlation between mean opinion score (MOS) and video quality assessment (VQA) metrics, including video bitrate, SSIM [17] and Video

---

[1]https://github.com/thu-media/Comyco

Table 1: Perfomance Comparison of QoE Models on Waterloo Streaming SQoE-III [12]

QoE model	Type	VQA	SRCC
Pensieve's [28]	linear	-	0.6256
MPC's [51]	linear	-	0.7143
Bentaleb's [9]	linear	SSIMplus [39]	0.6322
Duanmu's [12]	linear	-	0.7743
Comyco's	linear	VMAF [38]	**0.7870**

Multimethod Assessment Fusion (VMAF) [38], under the Waterloo Streaming QoE Database III (SQoE-III)[2] [12], where SSIM is a image quality metric which used by D-DASH [15] and VMAF is an objective full-reference video quality metric which is formulated by Netflix to estimate subjective video quality. Results are collected with Pearson correlation coefficient [8] as suggested by [5]. Experimental results (Fig. 6) show that VMAF achieves the highest correlation among all candidates, with the improvements in the coefficient of 16.39%-43.54%. Besides, VMAF are also a popular scheme with great potential on both academia and industry [3]. We, therefore, set $q(R_n) = \text{VMAF}(R_n)$.

QoE Parameters Setup. Recall that main goal of our paper is to propose a feasible ABR system instead of a convincing QoE metric. In this work, we attempt to leverage linear-regression methods to find the proper parameters. Specifically, we randomly divide the SQoE-III database into two parts, 80% of the database for training and 20% testing. We follow the idea by [12] and run the training process for 1,000 times to mitigate any bias caused by the division of data. As a result, we set $\alpha = 0.8469$, $\beta = 28.7959$, $\gamma = 0.2979$, $\delta = 1.0610$. We leverage spearman correlation coefficient (SRCC), as suggested by [12], to evaluate the performance of our QoE model with existing proposed models and the median correlation and its corresponding regression model are demonstrated in Table 1. As shown, QoE_v model outperforms recent work. In conclusion, the proposed QoE model is well enough to evaluate ABR schemes.

5.2 Video Datasets

To better improve the Comyco's generalization ability, we propose a video quality DASH dataset involves movies, sports, TV-shows, games, news and MVs. Specifially, we first collect video clips with highest resolution from Youtube, then leverage FFmpeg [14] to encode the video by H.264 codec and MP4Box [16] to *dashify* videos according to the encoding ladder of video sequences [1, 12]. Each chunk is encoded as 4 seconds. During the trans-coding process, for each video, we measure VMAF, VMAF-4K and VMAF-phone metric with the reference resolution of 1920 × 1080 respectively. In general, the dataset contains 86 complete videos, with 394,551 video chunks and 1,578,204 video quality assessments.

6 EVALUATION

6.1 Methodology

Virtual Player. We design a faithful ABR offline virtual player to train Comyco via network traces and video descriptions. The player

---

[2]SQoE-III is the *largest and most realistic dataset* for dynamic adaptive streaming over HTTP [12], which consists of a total of 450 streaming videos created from diverse source content and diverse distortion patterns.

is written in C++ and Python3.6 and is closely refers to several state-of-the-art open-sourced ABR simulators including Pensieve, Oboe and Sabre [43].

Testbed. Our work consists of two testbeds. Both server and client run on the 12-core, Intel i7 3.7 GHz CPUs with 32GB RAM running Windows 10. Comyco can be trained efficiently on both GPU and CPU. Detailing the testbed, that includes:

▷ **Trace-driven emulation.** Following the instructions of recent work [6, 28], we utilize Mahimahi [34] to emulate the network conditions between the client (ChromeV73) and ABR server (SimpleHTTPServer by Python2.7) via collected network traces.

▷ **Real world Deployment.** Details are illustrated in §6.5.

Network Trace Datasets. We collect about 3,000 network traces, totally 47 hours, from public datasets for training and testing:

▷ **Chunk-level network traces:** including HSDPA [41]: a well-known 3G/HSDPA network trace dataset, we use a slide-window to upsampling the traces as mentioned by Pensieve (1000 traces, 1s granularity); FCC [40]: a broadband dataset (1000 traces, 1s granularity); Oboe [48] (428 traces, 1-5s granularity): a trace dataset collected from wired, WiFi and cellular network connections (Only for validation.)

▷ **Synthetic network traces:** uses a Markovian model where each state represented an average throughput in the aforementioned range[28]. We create network traces in over 1000 traces with 1s granularity.

ABR Baselines. In this paper, we select several representational ABR algorithms from various type of fundamental principles:

▷ **Rate-based Approach (RB) [23]:** uses harmonic mean of past five throughput measured as future bandwidth.

▷ **BOLA [44]:** turns the ABR problem into a utility maximization problem and solve it by using the Lyapunov function. It's a buffer-based approach. We use BOLA provided by the authors [43].

▷ **Robust MPC [51]:** inputs the buffer occupancy and throughput predictions and then maximizes the QoE by solving an optimization problem. We use C++ to implement *RobustMPC* and leverage QoE_v (§5.1) to optimize the strategy.

▷ **Pensieve [28]:** the state-of-the-art ABR scheme which utilizes Deep Reinforcement Learning (DRL) to pick bitrate for next video chunks. We use the scheme implemented by the authors [27] but retrain the model for our work (§6.2).

6.2 Comyco vs. ABR schemes

In this part, we attempt to compare the performance of Comyco with the recent ABR schemes under several network traces via the trace-driven virtual player. The details of selected ABR baselines are described in §6.1. We use *EnvivoDash3*, a widely used [6, 28, 36, 51] reference video clip [1] and QoE_v to measure the ABR performance.

▷ ***Pensieve Re-training.*** We retrain Pensieve via our datasets (§6.1), NN architectures (§4.1) and QoE metrics (§5.1). Followed by recent work [6], our experiments use different entropy weights in the range of 5.0 to 1.0 and dynamically decrease the weight every 1000 iterations. Training time takes about 8 hours and we show that Pensieve outperforms RobustMPC, with an overall average QoE improvement of 3.5% across all sessions. Note that same experiments can improve the QoE_{lin} [51] by 10.5%. It indicates that QoE_v
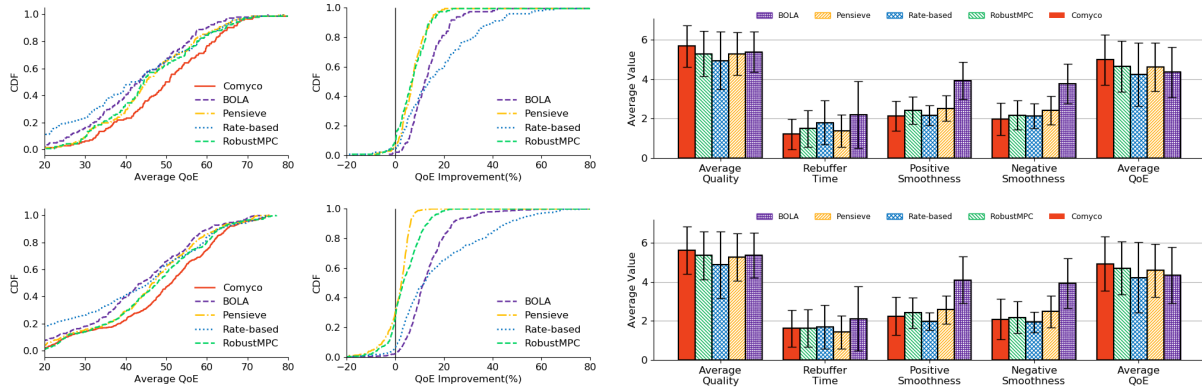
Figure 7: Comparing Comyco with existing ABR approaches under the HSDPA and FCC network traces. Results are illustrated with CDF distributions, QoE improvement curves and the comparion of several undelying metrics (§5.1).

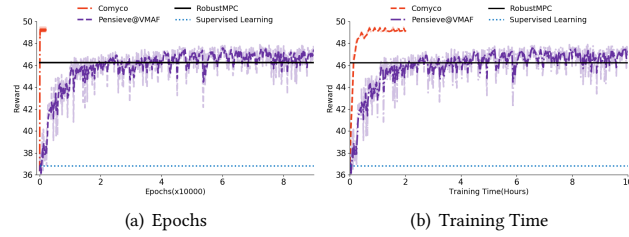(a) Epochs   (b) Training Time

Figure 8: Comparing the performance of Comyco with Pensieve and Supervised learning-based method under the HSDPA dataset. Comyco is able to achieve the highest performance with significant gains in sample efficiency.

cannot be easily improved because the metric reflects the real world MOS score.

Comparison of Learning-based ABR schemes. Figure 8 illustrates the average QoE of learning-based ABR schemes on HSDPA datasets. We validate the performance of two schemes respectively during the training. Results are shown with two perspectives including Epoch-Average QoE and Training time-Average QoE and we see about **1700x** improvement in terms of the number of samples required and about **16x** improvement in terms of training time required. As expected (§3.1), we observe that supervised learning-based method fails to find a strategy, which thereby leads to the poor performance.

Comyco vs. Existing ABRs. Figure 7 shows the comparison of QoE metrics for existing ABR schemes (§6.1). Comyco outperforms recent ABRs, with the improvements on average QoE of 7.5% - 17.99% across the HSDPA dataset and 4.85%-16.79% across the FCC dataset. Especially, Besides, we also show the CDF of the percentage of improvent in QoE for Comyco over existing schemes. Comyco surpasses state-of-the-art ABR approach Pensieve for 91% of the sessions across the HSDPA dataset and 78% of the sessions across the FCC dataset. What's more, we also report the performance of underlying metrics including average video quality (VMAF), rebuffering time, positive and negative smoothness, as well as QoE. We find that Comyco is well behaved on the average quality metric, which improves 6.84%-15.64% compared with other ABRs. Moreover,
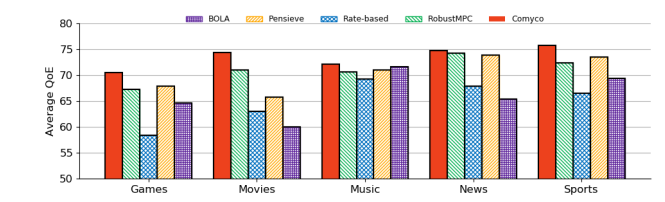

Figure 9: Comparing Comyco with existing ABR approaches under the Oboe network traces and various types of videos.

Comyco is able to avoid rebuffering and bitrate changes, which performs as same as state-of-art schemes.

6.3 Comyco with Multiple Videos

To better understand how does Comyco perform on various videos, we randomly pick videos from different video types and utilize Oboe network traces to evaluate the QoE_v performances of the proposed methods. Oboe network traces have diversity network conditions, which brings more challenges for us to improve the performance. Figure 9 illustrates the comparison of QoE metrics for state-of-the-art ABR schemes under various video types. We find that Comyco generalizes well under all considered video scenarios, with the improvements on average QoE of 2.7%-23.3% compared with model-based ABR schemes and 2.8%-13.85% compared with Pensieve. Specifically, Comyco can provide high quality ABR services under movies, news, and sports, which are all the scenarios with frequent scene switches. We also find that Comyco fails to demonstrate overwhelming performance in serving music videos. It's really an interesting topic and we'll discuss it in future work.

6.4 Ablation Study

In this section, we set up several experiments that aim to provide a thorough understanding of Comyco, including its hyperparameters and overhead. Note that, we have computed the offline-optimal results via dynamic programming and complete network status [28] before the experiment and treated it as a baseline.
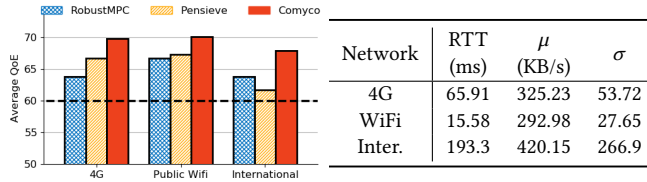
Comparison of different future step N. We report normalized QoE and raw time span of Comyco with different N and replay

Table 2: Comyco with different N and replay strategies.

$\alpha = 0.001/N$	5	6	7	8	9
Replay Off	0.883	0.893	0.917	0.932	0.942
Replay On	0.911	0.921	0.937	0.946	**0.960**
TimeSpan(Opt. Off)(ms)	1.56	8.74	58.44	389.68	2604.46

Table 3: Comyco with different α.

α	0.1	0.01	0.001	0.0001	0
k=4	0.883	0.895	**0.904**	0.881	0.867

Network	RTT (ms)	μ (KB/s)	σ
4G	65.91	325.23	53.72
WiFi	15.58	292.98	27.65
Inter.	193.3	420.15	266.9

Figure 10: Comparing Comyco with Pensieve and Ro-bustMPC under the real-world network conditions. We take $QoE = 60$ as baselines.

experience strategy in Table 2. Results are collected under the Oboe dataset. As shown, we find that experience replay can help Comyco learn better. Despite the outstanding performance of Comyco with N=9, this scheme lacks the algorithmic efficiency and can hardly be deployed in practice. Thus, we choose k=8 for harmonizing the performance and the cost.

Comyco with different α. Further, we compare the normalized QoE of Comyco with different α under the Oboe dataset. As listed in Table 3, we confirm that $\alpha = 0.001$ represents the best parameters for our work. Meanwhile, results also prove the effective of utilizing entropy loss (§4.3).

Comyco Overhead. We calculate [33] the number of floating-point operations (FLOPs) of Comyco and find that Comyco has the computation of 229 Kflops, which is only 0.15% of the light-weighted neural network ShuffleNet V2 [26] (146 Mflops). In short, we believe that Comyco can be successfully deployed on the PC and laptop, or even, on the mobile.

6.5 Comyco in the Real World

We establish a full-system implementation to evaluate Comyco in the wild. The system mainly consists of a video player, an ABR server and an HTTP content server. On the server-side, we deploy an HTTP video content Server. On the client-side, we modify Dash.js [1] to implement our video player client and we use Chrome to watch the video. Moreover, we implement Comyco as a service on the ABR server. We evaluate the performance of proposed schemes under various network conditions including 4G/LTE network, WiFi network and international link (from Singapore to Beijing). Figure 10 illustrates network status, where μ is the average throughput measured and σ represents standard deviation from the average. For each round, we randomly picks a scheme from candidates and summarize the bitrate selected and rebuffering time

for each chunk. Each experiment takes about 2 hours. Figure 10 shows the average QoE results for each scheme under different network conditions. It's clear that Comyco also outperforms previous state-of-the-art ABR schemes and it improves the average QoE of 4.57%-9.93% compared with Pensieve and of 6.43%-9.46% compared with RobustMPC.

7 RELATED WORK

ABR schemes. Client-based ABR algorithms [10] are mainly organized into two types: model-based and learning-based.

Model-based. The development of ABR algorithms begins with the idea of predicting throughput. FESTIVE [23] estimates future throughput via the harmonic mean of the throughput measured for the past chunk downloads. Meanwhile, many approaches are designed to select the appropriate high bitrate next video chunk and avoid rebuffering events based on playback buffer size observed. BBA [21] proposes a linear criterion threshold to control the available playback buffer size. Mixed approaches, e.g., MPC [51], select bitrate for the next chunk by adjusting its throughput discount factor based on past prediction errors and estimating its playback buffer size. What's more, Akhtar et al. [6] propose an auto-tuning method to improve the model-based ABR's performance.

Learning-based: Several attempts have been made to optimize the ABR algorithm based on RL method due to the difficulty of tuning mixed approaches for handling different network conditions. Pensieve [28] is a system that uses DRL to select bitrate for future video chunks. D-DASH [15] uses Deep Q-learning method to perform a comprehensive evaluation based on state-of-the-art algorithms. Tiyuntsong optimizes itself towards a rule or a specific reward via the competition with two agents under the same network condition [18].

Imitation Learning meets Networking. Imitation learning [22] has been widely used in the various fields including networking. Tang et al. [46] propose real-time deep learning based intelligent network traffic control method to represent the considered Wireless Mesh Network (WMN) backbone via imitation learning. Indigo [50] uses DAgger [42] to train a congestion-control NN scheme in the offline network emulator.

8 CONCLUSION

In this work, we propose Comyco, a learning-based ABR system which aim to thoroughly improve the performance of learning-based algorithm. To overcome the sample inefficiency problem, we leverage imitation learning method to *guide* the algorithm to explore and exploit the *better* policy rather than stochastic sampling. Moreover, we construct the video quality-based ABR system, including its NN architectures, datasets and QoE metrics. With trace-driven emulation and real-world deployment, we show that Comyco significantly improves the performance and effectively accelerates the training process.

REFERENCES

[1] 2019. DASH Industry Forum | Catalyzing the adoption of MPEG-DASH. (2019). https://dashif.org/

[2] 2019. HTTP Live Streaming. https://developer.apple.com/streaming/. (2019).

[3] Anne Aaron, Zhi Li, Megha Manohara, Joe Yuchieh Lin, Eddy Chi-Hao Wu, and C-C Jay Kuo. 2015. Challenges in cloud based ingest and encoding for high quality streaming media. In *2015 IEEE International Conference on Image Processing (ICIP)*. IEEE, 1732–1736.

[4] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. TensorFlow: A System for Large-Scale Machine Learning.. In *OSDI*, Vol. 16. 265–283.

[5] Tasnim Abar, Asma Ben Letaifa, and Sadok El Asmi. 2017. Machine learning based QoE prediction in SDN networks. In *2017 13th International Wireless Communications and Mobile Computing Conference (IWCMC)*. IEEE, 1395–1400.

[6] Zahaib Akhtar and et al. 2018. Oboe: auto-tuning video ABR algorithms to network conditions. In *SIGCOMM 2018*. ACM, 44–58.

[7] David M Beazley et al. 1996. SWIG: An Easy to Use Tool for Integrating Scripting Languages with C and C++.. In *Tcl/Tk Workshop*. 43.

[8] Jacob Benesty, Jingdong Chen, Yiteng Huang, and Israel Cohen. 2009. Pearson correlation coefficient. In *Noise reduction in speech processing*. Springer, 1–4.

[9] Abdelhak Bentaleb, Ali C Begen, and Roger Zimmermann. 2016. SDNDASH: Improving QoE of HTTP adaptive streaming using software defined networking. In *Proceedings of ACM MultiMedia 2016*. ACM, 1296–1305.

[10] Abdelhak Bentaleb, Bayan Taani, Ali C Begen, Christian Timmerer, and Roger Zimmermann. 2018. A Survey on Bitrate Adaptation Schemes for Streaming Media over HTTP. *IEEE Communications Surveys & Tutorials* (2018).

[11] Cisco. 2017. Cisco Visual Networking Index: Forecast and Methodology, 2016-2021. (2017). https://www.cisco.com/c/dam/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.pdf

[12] Zhengfang Duanmu, Abdul Rehman, and Zhou Wang. 2018. A Quality-of-Experience Database for Adaptive Video Streaming. *IEEE Transactions on Broadcasting* 64, 2 (June 2018), 474–487.

[13] Zhengfang Duanmu, Kai Zeng, Kede Ma, Abdul Rehman, and Zhou Wang. 2017. A quality-of-experience index for streaming video. *IEEE Journal of Selected Topics in Signal Processing* 11, 1 (2017), 154–166.

[14] FFmpeg. [n. d.]. FFmpeg. ([n. d.]). http://ffmpeg.org/

[15] M. Gadaleta, F. Chiariotti, M. Rossi, and A. Zanella. 2017. D-DASH: A Deep Q-Learning Framework for DASH Video Streaming. *IEEE Transactions on Cognitive Communications and Networking* 3, 4 (Dec 2017), 703–718. https://doi.org/10.1109/TCCN.2017.2755007

[16] GPAC. [n. d.]. MP4BOX. ([n. d.]). https://gpac.wp.imt.fr/mp4box/

[17] Alain Hore and Djemel Ziou. 2010. Image Quality Metrics: PSNR vs. SSIM. (2010), 2366–2369.

[18] Tianchi Huang, Xin Yao, Chenglei Wu, Rui-Xiao Zhang, and Lifeng Sun. 2018. Tiyuntsong: A Self-Play Reinforcement Learning Approach for ABR Video Streaming. *arXiv preprint arXiv:1811.06166* (2018).

[19] Tianchi Huang, Rui-Xiao Zhang, Chao Zhou, and Lifeng Sun. 2018. QARC: Video Quality Aware Rate Control for Real-Time Video Streaming based on Deep Reinforcement Learning. In *2018 ACM Multimedia Conference on Multimedia Conference*. ACM, 1208–1216.

[20] Te-Yuan Huang, Chaitanya Ekanadham, Andrew J. Berglund, and Zhi Li. 2019. Hindsight: Evaluate Video Bitrate Adaptation at Scale. In *Proceedings of the 10th ACM Multimedia Systems Conference (MMSys '19)*. ACM, New York, NY, USA, 86–97. https://doi.org/10.1145/3304109.3306219

[21] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. 2015. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. *ACM SIGCOMM Computer Communication Review* 44, 4 (2015), 187–198.

[22] Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. 2017. Imitation learning: A survey of learning methods. *ACM Computing Surveys (CSUR)* 50, 2 (2017), 21.

[23] Junchen Jiang, Vyas Sekar, and Hui Zhang. 2014. Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive. *TON* 22, 1 (2014), 326–340.

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

[25] Michael Laskey, Jonathan Lee, Roy Fox, Anca Dragan, and Ken Goldberg. 2017. Dart: Noise injection for robust imitation learning. *arXiv preprint arXiv:1703.09327* (2017).

[26] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. 2018. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 116–131.

[27] Mao. 2017. hongzimao/pensieve. (Jul 2017). https://github.com/hongzimao/pensieve

[28] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. 2017. Neural adaptive video streaming with pensieve. In *Proceedings of the 2017 ACM SIGCOMM Conference*. ACM, 197–210.

[29] Hongzi Mao, Shaileshh Bojja Venkatakrishnan, Malte Schwarzkopf, and Mohammad Alizadeh. 2019. Variance Reduction for Reinforcement Learning in Input-Driven Environments. *international conference on learning representations* (2019).

[30] Russell Mendonca, Abhishek Gupta, Rosen Kralev, Pieter Abbeel, Sergey Levine, and Chelsea Finn. 2019. Guided Meta-Policy Search. *arXiv preprint arXiv:1904.00956* (2019).

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

[32] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[33] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. 2016. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440* (2016).

[34] Ravi Netravali, Anirudh Sivaraman, Somak Das, Ameesh Goyal, Keith Winstein, James Mickens, and Hari Balakrishnan. 2015. Mahimahi: accurate record-and-replay for HTTP. (2015), 417–429.

[35] Takayuki Osa, Joni Pajarinen, Gerhard Neumann, J Andrew Bagnell, Pieter Abbeel, Jan Peters, et al. 2018. An algorithmic perspective on imitation learning. *Foundations and Trends® in Robotics* 7, 1-2 (2018), 1–179.

[36] Pablo Gil Pereira, Andreas Schmidt, and Thorsten Herfet. 2018. Cross-Layer Effects on Training Neural Algorithms for Video Streaming. In *Proceedings of the 28th ACM SIGMM Workshop on Network and Operating Systems Support for Digital Audio and Video*. ACM, 43–48.

[37] Yanyuan Qin, Shuai Hao, Krishna R Pattipati, Feng Qian, Subhabrata Sen, Bing Wang, and Chaoqun Yue. 2018. ABR streaming of VBR-encoded videos: characterization, challenges, and solutions. In *Proceedings of CoNeXT 2018*. ACM, 366–378.

[38] Reza Rassool. 2017. VMAF reproducibility: Validating a perceptual practical video quality metric. In *Broadband Multimedia Systems and Broadcasting (BMSB), 2017 IEEE International Symposium on*. IEEE, 1–2.

[39] Abdul Rehman, Kai Zeng, and Zhou Wang. 2015. Display device-adapted video quality-of-experience assessment. In *Human Vision and Electronic Imaging XX*, Vol. 9394. International Society for Optics and Photonics, 939406.

[40] Measuring Fixed Broadband Report. 2016. Raw Data Measuring Broadband America 2016. https://www.fcc.gov/reports-research/reports/measuring-broadband-america/raw-data-measuring-broadband-america-2016. (2016). [Online; accessed 19-July-2016].

[41] 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*. ACM, 114–118.

[42] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. 2011. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. 627–635.

[43] 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 MMSys*. ACM, 123–137.

[44] Kevin Spiteri, Rahul Urgaonkar, and Ramesh K Sitaraman. 2016. BOLA: Near-optimal bitrate adaptation for online videos. In *INFOCOM 2016, IEEE*. IEEE, 1–9.

[45] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

[46] F. Tang, B. Mao, Z. M. Fadlullah, N. Kato, O. Akashi, T. Inoue, and K. Mizutani. 2018. On Removing Routing Protocol from Future Wireless Networks: A Real-time Deep Learning Approach for Intelligent Traffic Control. *IEEE Wireless Communications* 25, 1 (February 2018), 154–160. https://doi.org/10.1109/MWC.2017.1700244

[47] Yuan Tang. 2016. TF. Learn: TensorFlow's high-level module for distributed machine learning. *arXiv preprint arXiv:1612.04251* (2016).

[48] Usc-Nsl. 2018. USC-NSL/Oboe. (Oct 2018). https://github.com/USC-NSL/Oboe

[49] Zhou Wang. 2017. Video QoE: Presentation Quality vs. Playback Smoothness. (Jul 2017). https://www.ssimwave.com/science-of-seeing/video-quality-of-experience-presentation-quality-vs-playback-smoothness/

[50] Francis Y Yan, Jestin Ma, Greg D Hill, Deepti Raghavan, Riad S Wahby, Philip Levis, and Keith Winstein. 2018. Pantheon: the training ground for Internet congestion-control research. In *2018 {USENIX} Annual Technical Conference ({USENIX} {ATC} 18)*. 731–743.

[51] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. 2015. A control-theoretic approach for dynamic adaptive video streaming over HTTP. In *ACM SIGCOMM Computer Communication Review*. ACM, 325–338.