Quality-aware Neural Adaptive Video Streaming with Lifelong Imitation Learning

Tianchi Huang, Chao Zhou, Xin Yao, Rui-Xiao Zhang, Chenglei Wu, Bing Yu, Lifeng Sun

***Abstract*—Existing Adaptive Bitrate (ABR) algorithms pick future video chunks' bitrates via fixed rules or offline trained models to ensure good quality of experience (QoE) for Internet video. Nevertheless, data analysis demonstrates that a good ABR algorithm is required to continually and fast update for adapting itself to time-varying network conditions. Therefore, we propose Comyco, a video quality-aware learning-based ABR approach that enormously improves recent schemes by i) picking the chunk with higher perceptual video qualities rather than video bitrates; ii) training the policy via imitating expert trajectories given by the expert strategy; iii) employing the lifelong learning method to continually train the model w.r.t the fresh trace collected by the users. To achieve this, we develop a complete quality-aware lifelong imitation learning-based ABR system, construct quality-based neural network architecture, collect a quality-driven video dataset, and estimate QoE metrics with video quality features. Using trace-driven and real-world experiments, we demonstrate Comyco reaches 1700× improvements in the number of samples required and 16× speedup in the training time compared with the prior work. Meanwhile, Comyco outperforms existing methods, with the improvements on average QoE of 7.5%-16.79%. Moreover, experimental results on continual training also illustrate that lifelong learning helps Comyco further improve the average QoE of 1.07%-9.81% in comparison to the offline trained model.**

***Index Terms*—Imitation Learning, Quality-aware, Lifelong Learning, Adaptive Video Streaming.**

I. Introduction

Recent years have witnessed a tremendous increase in the requirements of watching online videos [1]. Adaptive bitrate (ABR) streaming, the method that dynamically controls the video player to download different bitrate video for the next chunk, has become a leading scheme to deliver video streaming services with high quality of experience (QoE) to the users [2]. Recently, ABR technologies have been widely used by YouTube [3], Netflix [4], and iQiyi [5]. Existing model-based ABR approaches (§VIII) pick the next chunk's video bitrate via only current network status [6], [7], or buffer occupancy [8], [9], or joint consideration of these two factors [10], [11]. However, such heuristics are usually set up with presumptions, that fail to work well under unexpected network conditions [12]. Thus, several attempts, i.e., learning-based ABR algorithms, have been made to adopt reinforcement learning (RL) [12]–[14] or self-learning method [15] to *generalize* the strategies without any presumptions, and thus, providing a feasible method to solve the ABR task from another perspective.

While previous work has demonstrated considerable QoE improvement in a different manner, in this study, we attempt to understand whether current ABR methods have already been satisfied with nowadays' network (§II). We, therefore, collect a large corpus of network traces (Kwai dataset) on the leading video streaming platform Kuaishou [16] (§II-B). The analysis shows that 1) more than 80% of network traces require an adaptive streaming method to ensure high QoE. 2) learning-based ABR approach (i.e., Pensieve [12]) is often required to be trained from scratch for over 4 hours. However, the network distribution has changed dramatically during the time of training convergence. As a result, the algorithm, trained on past network scenarios, may hardly provide comparable performances under the current network condition. 3) as much as the overall network condition shows different throughput distribution at large time intervals, it changes slowly and smoothly with time. Hence, learning-based ABR algorithms should be updated effectively and efficiently for smoothing the vibration of network conditions. To achieve this goal, we summarize the challenges from the following perspectives:

▷ *How to implement a quality-aware ABR system?* The majority of existing ABR approaches [10], [12], [17] place less importance on the video quality information, while perceptual video quality is a non-trivial feature for evaluating QoE (§V-A, [18]). Consequently, even though these schemes have achieved higher QoE objectives, they may generate the strategy diverging from the actual demand. (§III-A)

▷ *How to empower the training efficiency for learning-based ABR algorithms?* Recent Reinforcement Learning (RL)-based ABR schemes [12], [13] lack the efficiency of both collected and exploited expert samples, which leads to the inefficient training [19]. (§III-B)

▷ *How to achieve continual learning for the ABR system?* Learning-based ABR methods should be incrementally updated with *fresh* network traces, and in the meanwhile, the selected traces should be less but critical enough to represent bandwidth distributions of the current network. (§III-C)

We find an opportunity to address the last two issues in real-world network environments by leveraging the concept of lifelong imitation learning. On the one hand, imitation learning enables the ABR system to achieve fast training. On the other hand, a lifelong learning method allows the neural network (NN) to continually integrate the evolution in network distributions into the passing time. Meanwhile, quality-aware learning-based ABR algorithm is still challenging since the

T. Huang, X. Yao, RX Zhang, C. Wu, and L. Sun are with the Department of Computer Science and Techonology, Tsinghua University, Beijing, 100084, China. (e-mail: {htc19, yaox16, zhangrx17, wucl18}@mails.tsinghua.edu.cn, sunlf@tsinghua.edu.cn)

C. Zhou, and B. Yu, are with Beijing Kuaishou Technology Co., Ltd, Beijing, China. (e-mail: {zhouchao, yubing}@kuaishou.com)

✉ Lifeng Sun, Chao Zhou are the corresponding authors. (e-mail: sunlf@tsinghua.edu.cn, zhouchao@kuaishou.com)

state-of-the-art learning-based scheme [12], [15] lacks almost all the modules of constructing a quality-aware ABR system, that includes, viable neural network models, feasible and high-efficiency training methodologies, dedicated video datasets based on video quality metrics, and quality-based QoE methods.

Following this insight, we propose *Comyco*, a novel video quality-aware lifelong imitation learning-based ABR system, aiming to remarkably improve the overall performance of ABR algorithms via tackling the above challenges. Different from previous work [12], Comyco is quality-aware and mainly composed of the inner-loop system and the outer-loop system, and is equipped with the following properties (§IV):

▷ *Comyco aims to select bitrate with high perceptual video quality rather than high video bitrate.* To achieve this goal, we first integrate the information of video contents, network status, and video playback states into the Comyco's NN for bitrate selection (§IV-A1). Next, we use VMAF [20], a state-of-the-art machine learning-based objective full-reference perceptual video quality metric, to measure the video quality. Meanwhile, we propose a linearity quality-based QoE metric that achieves the state-of-art performance on Waterloo Streaming SQoE-III [21] dataset (§V-A). Finally, we collect a DASH-video dataset with various types of videos (§V-B).

▷ *Comyco utilizes the inner-loop system (§IV-A), which leverages imitation learning [22] for training the neural network (NN).* Since the near-optimal policy can be precisely and instantly estimated via the current state in the ABR scenario, the collected expert policies can enable the NN for fast learning. The agent is allowed to *explore* the environment and *learn* the policy via the expert policies given by the solver.

▷ *Comyco adopts the outer-loop system (§IV-B) to achieve continual learning.* We consider the process of continuous adaptation to network status as a lifelong learning process. The key idea is to filter out the useful traces collected from the client, and periodically update the NN via the inner-loop system and learn the strategies using Learning without Forgetting (LwF) [23] method.

Furthermore, we evaluate Comyco's inner-loop and outer-loop system via trace-driven and real-world experiments. Using trace-driven emulation (§VI-A2), we find that Comyco significantly accelerates the training process, with 1700× improvements in terms of a number of samples required compared to the recent work (§VI-A3), and 16× speedup on the training time. Comparing with existing schemes, Comyco outperforms them under various network conditions (§VI-A2) and videos (§V-B), 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 [12], with the improvements on the average video quality of 7.37% under the same rebuffering time. Further, we report results that highlight Comyco's performance with different hyper-parameters and settings (§VI-A5). Extensive results over the real-world network scenarios indicate the superiority of Comyco over existing state-of-the-art approaches (§VI-A6). Moreover, we also discuss the performance of the outer-loop system, namely lifelong Comyco (§VI-B). Experimental results demonstrate that the using outer-loop system can effectively help the proposed method further improve the average QoE of 1.07%-9.81% compared with the offline fixed model (§VI-B3). Meanwhile, we further analyze the performance comparison of Comyco and state-of-the-art model-based ABR approach RobustMPC [10], and we find that lifelong Comyco can automatically adapt to the complicated environment and stochastic property in various network conditions. The comparison between lifelong Comyco and the online optimal policy illustrates that the proposed scheme has almost achieved the near-optimal performance (§VI-B5).

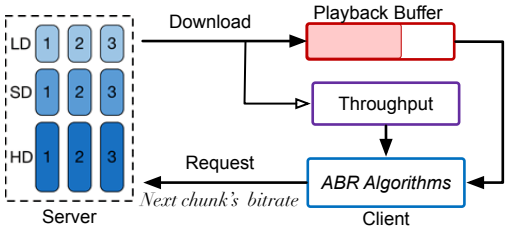


Fig. 1. An overview of HTTP adaptive video streaming. The system is comprised of a video server and a video client. ABR, placed on the client side, is an algorithm that determines next chunks' bitrates w.r.t past throughput and current buffer occupancy.

Contribution. We summarize the contributions as follows:

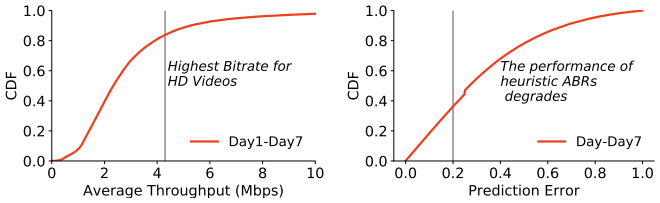
- Using data-driven analysis, we identify the short-comings of today's ABR schemes and propose Comyco, a video quality-aware lifelong learning-based ABR system, that significantly ameliorates the weakness of the learning-based ABR schemes from several perspectives (§III).
- Unlike prior work, Comyco picks the video chunk with high perceptual video qualities instead of high video bitrates. Experiments results also demonstrate the superiority of our proposed algorithm (§III-A,§IV).
- To the best of our knowledge, we are the first to leverage imitation learning to accelerate the training process for ABR tasks. Results show that exploring imitation learning can not only achieve sample efficiency but also improve the overall performance (§IV-A,§VI-A).
- We consider the continuous updating task of ABR as a lifelong learning process. Results demonstrate that adopting lifelong learning enables the ABR algorithm to effectively adapt the time-vary network conditions (§IV-B,§VI-B).

II. Background and Motivation

In this section, we begin by introducing ABR's background. Then we analyze today's ABR services. Finally, we highlight the limitations of strawman solutions for ABR schemes without lifelong learning and present key insights that lead to implementing a new ABR system for providing better QoE to the users at any time.

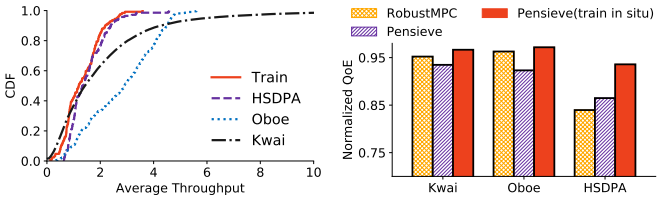
A. ABR Overview

Due to the rapid development of network services, watching videos online has already become a common trend. Today, the predominant form for video delivery is adaptive video streaming, such as HLS (HTTP Live Streaming) [24] and DASH [25], which is a method that dynamically selects



(a) CDF of session throughput. (b) CDF of prediction error.

Fig. 2. An overview of Kwai dataset, including the distribution of average throughput and throughput's prediction error using popular throughput prediction method [7].



(a) CDF of session throughput on different network traces (b) Normalized QoE on the selected network environments

Fig. 3. Comparing the throughput distribution on different network datasets. We also report normalized QoE on each network dataset. It's notable that Pensieve's training time lasts about 8 hours.

video bitrates according to network conditions and clients' buffer occupancy. As shown in Figure 1, the traditional video streaming framework consists of a video player client with a constrained buffer length and an HTTP-Server or Content Delivery Network (CDN) [7]. 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 the 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 events [2].

B. Analysis for Today's ABR Services

Our work starts with a realistic problem: *with the rapid improvements of today's network bandwidth, are ABR algorithms still necessary for video streaming services to provide better QoE to the users?* To answer this question, we require a *fresh* throughput dataset in large-scale, continuous throughput measurements, and long session duration. However, revisiting previously proposed public throughput trace datasets [17], [26], [27], we observe that such existing datasets lack either the diversity of throughput traces or the continuous measurement through the entire weeks, which finally unable to use them directly for research purpose [1]. To this end, we collect a large-scale network bandwidth dataset, namely Kwai, from the video streaming viewers of Kuaishou [16]. Kuaishou is a leading video streaming platform in China that has over 300 million users worldwide. The dataset consists of over 86 thousand traces from 9,941 users, 7 days in total (1104 hours in terms of overall bandwidth time recorded.) from various network conditions collected in June 2019. Then we utilize the Kwai dataset to implement several experiments for answering the questions above and dedicate several observations.

▷ `Observation1`. *Experiments illustrate that ABR algorithms are still necessary for 80% of today's network conditions. Meanwhile, the state-of-the-art heuristic method MPC (Model Predictive Control) [10] only performs well under almost 40% of all sessions.*

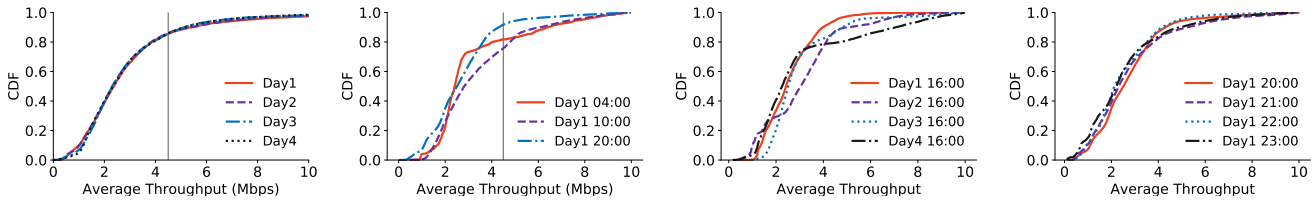
To better investigate the importance of ABR algorithms for today's network, we compute *average throughput* on all the sessions of the Kwai dataset and report them as the CDF distribution plot in Figure 2. Figure 2(a) shows, assuming that the highest video bitrate of per chunks is 4.3Mbps [2], we find that over 80% of sessions require ABR algorithms to adjust next chunk's bitrate for avoiding rebuffering events if the users prefer watching the video with the highest bitrates since the average bandwidth of the client is lower than the chunk with the highest bitrate. Moreover, recent work [10] demonstrates that MPC's performance heavily depends on the throughput accuracy. If the throughput's prediction error is over 20%, MPC will be performed under 85% of optimal QoE. Thus, we also illustrate the CDF distribution of prediction error in Figure 2(b), where the prediction error is computed as mean absolute percentage error (MAPE). Surprisingly, over 60% of sessions gain a large prediction error (over 20%), which means, existing heuristics fail to guarantee the performance of 40% sessions. To this end, we aim to use *learning-based ABR algorithm* rather than heuristics for achieving better QoE.

▷ `Observation2`. *Measurements show the network distribution will be different if the time gap lasts over 6 hours. However, the training time of recent learning-based ABR algorithms is in the range of 4-23 hours [12], [13]. Hence, the learned strategy (trained on previous network distributions) may perform poorly on current network conditions.*

Note that recent client-based ABRs often trained [12] or designed [10], [28] once and deployed on the users' client without any further changes. Such observation leads to another critical question: *can ABRs tame the complexity of dynamic network conditions without updating?* We, therefore, evaluate the performance of existing ABR algorithms (i.e., Pensieve [12], RobustMPC [10] and Pensieve (train in situ [29])) over different network conditions, including HSDPA, Oboe (§VI-A2) and Kwai, on the virtual player (§VI-A2). In detail, we train Pensieve on the training set, provided by the original Pensieve work [30] and validate it on the various network traces with the same trained model. In contrast, Pensieve (train in situ) means we train and validate Pensieve on the same network condition. Results on Figure 3(b) elaborate that 1) the overall performance of Pensieve heavily rely on the similarity of the throughput distribution between the training set and the validation network environments (see more in Figure 3(a)), and 2) learning Pensieve in situ always outperforms others

[1]It's notable that CS2P's network dataset [5] is fit for our work, but it still has not been published yet (Jan. 2020).

[2]It's a standard-setting for HD (1080p) videos [12], [17]



(a) Day. (b) Representative time. (c) Day with a designated time. (d) Continuous hours.

Fig. 4. CDF of session throughput by continuous day, representative time, specific time, as well as continuous time.

under all considered network environments. Hence, we believe that the learning-based ABR algorithm has plenty of room for improvement, e.g., narrowing the difference between the distribution of the training set and testing set.

So, does today's network emerge the same network distribution at different times? Figure 4(a) illustrates the CDF of the average throughput from June 1 to June 4. We find that there's no obvious difference between the throughput distribution on each day. Thus, we further discuss the throughput distribution with three representative times on the same day, where the time represents the sleeping time, working time, and resting time, respectively. Results are shown in Figure 4(b), and we can see that the network distribution is strongly correlated with human behavior. For example, the average throughput in the night (20:00) is lower than that in the morning (10:00). Technically, Pensieve takes over 4-8 hours (or 120-200 thousand iterations) to train a reliable strategy on current network distributions. Thus, although its training time is rather short compared with other learning-based ABRs [15], we believe that the current strategy, i.e., training a NN from scratch in 4 hours, doesn't always satisfy the users' requirements.

▷ `Observation3`. *ABR algorithms is allowed to be generated or updated efficiently within short duration since the bandwidth distribution evolves slowly over time, as the distribution of two adjacent time points (1 hour) is approximately the same.*

Figure 4(c) and 4(d) elaborate the CDF of throughput distribution from another perspective, and we can see that although the bandwidth distribution seems different at the same time on different days (Figure 4(c)), but the conditions of two adjacent time points (1 hour) is approximately the same (Figure 4(d)). To this end, the intuitive idea is to continually train the model with a short period, which allows the ABR algorithm to update dynamically instead of leveraging fixed parameters or rules. In our work, we set the training period as *1 hour*. In detail, we take 50-55 minutes for collecting network traces and use only 5-10 minutes for training the NN (§IV-B3).

▷ **Summary.** Exploring the aforementioned opportunities, however, requires a learning-based ABR scheme with not only achieving outperformed performances with fast efficiency training in a short period but also continually learning to fit the variety of network environments.

III. Challenges and Key Ideas

In this section, on the basis of the aforementioned observations, we mainly generalize three key challenges from several perspectives, that includes, how to develop a complete video quality-based ABR system (§III-A), how to train the NN via imitation learning (§III-B), and how to deploy the system for lifelong updating (§III-C).

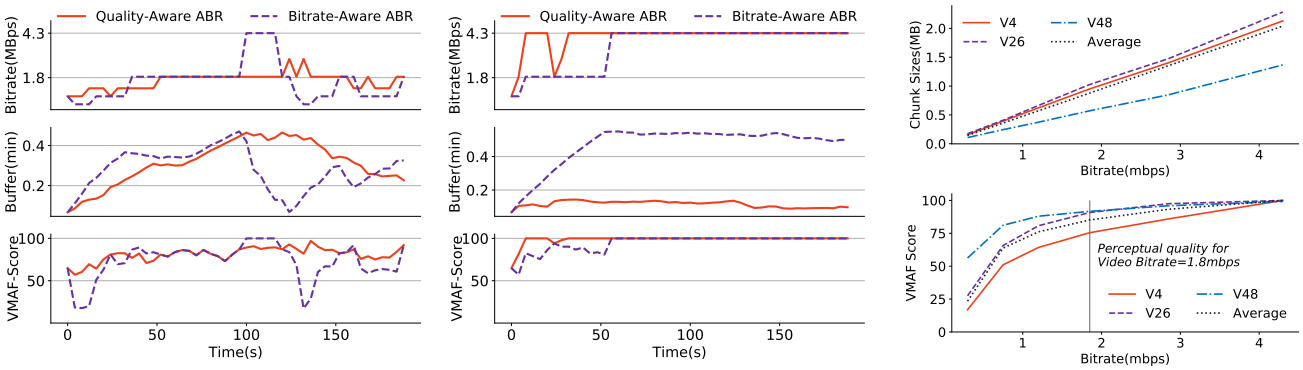
While previous work attempts to solve the ABR problem with different manner (§VIII), existing ABR schemes typically suffer from several issues. To that end, we summarize the key challenges as follows:

A. Challenges for Perceptual Video Quality-aware ABRs

Previous popular ABR schemes [5], [10], [12], [17] are often 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 [31]. Meanwhile, recent work [32], [33] has found that perceptual video quality features play a vital part in evaluating the performance of VBR-encoded ABR streaming services.

To better understand the difference between the quality-aware and the bitrate-aware ABR scheme, we report the trajectory generated by the two methods in Figure 5, in which the perceptual quality is measured by Video Multi-Method Assessment Fusion(VMAF) [20], a smart perceptual video quality assessment algorithm based on support vector machine(SVM), which currently stands for the state-of-the-art quality assessment metric [33]. More information please refer to §V-A. Figure 5(a) shows, the bitrate-aware method blindly selects the video chunk with higher bitrate but neglects the corresponding video quality. However, comparing the trajectory of bitrate-aware with the quality-aware approach, we find that the bitrate-aware method often downloads low-efficiency chunk. For instance, during the *playback time=14*, although the bitrates of the two choices are only one level different, the chunk of higher bitrates gains 243.44% (17.59→60.43) improvements on the perceptual video quality compared with that of the lower one (see more in Figure 5(c), *video chunk 4*). Meanwhile, the bitrate-aware method also wastes the buffer on achieving a slight increase in video quality, which may eventually cause unnecessary stalling events in the future. E.g., during the *playback time=100*, the bitrate-aware algorithm chooses the highest bitrates, but only gains 12.13% (89.13→99.97) in terms of the video quality compared with the quality-aware method (see more in Figure 5(c), *video chunk 26*). On the contrary, the quality-aware algorithm always picks the best-efficient chunk with high perceptual video quality and preserves the buffer occupancy within an allowable range.

Hence, one of the better solutions is to add video bitrates as another metric to describe the perceptual video quality.

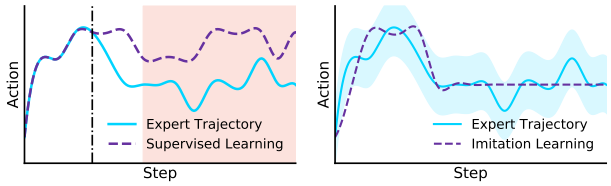


(a) Quality-aware ABR picks best-fit bitrates via the bitrate-quality curve on each chunk.

(b) Our approach can effectively utilize the buffer occupancy compared with the existing scheme.

(c) The correlation between the bitrate and the chunk size, as well as the bitrate and the VMAF.

Fig. 5. We evaluate quality-aware ABR algorithm and bitrate-aware ABR algorithm with the same video over HSDPA [27] network traces respectively. Results are plotted as the curves of selected bitrate, buffer occupancy and the selected chunk's VMAF (§V-A, [20]) for entire sessions.



(a) Supervised learning

(b) Imitation learning

Fig. 6. 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.

Nevertheless, current approaches, especially learning-based ABRs [12], lack fundamental quality-based settings, i.e., architectures, metrics, datasets, and so on. We, therefore, encounter the first challenge of our work: *How to construct a video quality-aware ABR system?*

Our Solution. In this paper, our solution 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 (§IV-A1). 2) We propose a quality-based QoE metric (§V-A). 3) We collect a video quality DASH dataset which includes various types of videos (§V-B).

B. Challenges for Sample Efficiency

Recent learning-based ABR schemes adopt RL methods to maximize the average QoE objectives. 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 [22], [34]. Thus, we are facing the second challenge: *Considering the characteristic of ABR tasks, can we precisely estimate the optimal direction of gradients to guide the model for better updating?*

Our solution. 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 does not really know the optimal strategy [35]. However, recent work [10]–[12], [17] has demonstrated that the ABR process can be precisely emulated by an offline virtual player (§VI-A2) 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 (§IV-A2). Thus, the intuitive idea is to leverage supervised learning methods to minimize the loss between the predicted and the expert policy. Nevertheless, it's impractical since the off-policy method [35] suffers from *compounding error* when the algorithm executes its policy, leading it to drift to new and unexpected states [36]. For example, as shown in Figure 6(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 transited 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 (the red area in the figure). As a result, supervised-learning methods lack the ability to 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 [22]. Imitation learning method allows 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 $\ell_t(\pi_t, \pi_t^*)$ 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 $\ell_t(\pi_{t+1}, \pi_{t+1}^*)$. Thus, for each π_t in the class of policies $T \in \{\pi_0, \ldots, \pi_t\}$, we can find the policy $\hat{\pi}$ through any supervised learning algorithms (Eq. 1).

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

Figure 6(b) elaborates the principle of imitation learning-

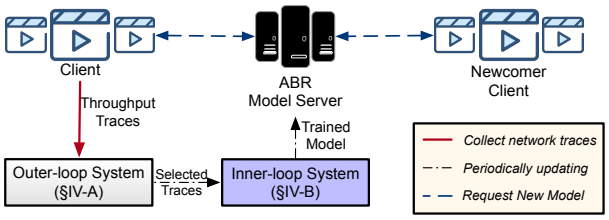


Fig. 7. Comyco System Overview. The system is composed of inner-loop system and outer-loop system.

based ABR schemes: the algorithm attempts to explore the strategy in a range near the expert trajectory to avoid compounding errors. Moreover, Figure 5(b) also shows that the imitation learning method can also help taming the complexity of the ABR task, keeping the buffer occupancy within a low but safe range.

C. Challenges for Lifelong Updating

Moreover, previous observation shows that recent learning-based ABRs fail to deploy in the real-world scenarios since such methods are required to online updating efficiently for overcoming the time-varies of network conditions. At the same time, although we've already attempted to leverage imitation learning rather than reinforcement learning for fast updating, such methods still suffer from the large corpus of network traces on each period, and finally, resulting in the failure of converge within an acceptable time. Hence, we finally list the third challenge: *Based on previously trained model, is there any possibility of incrementally train an ABR algorithm with a succinct yet efficient group of network traces?*

Our Solution. We consider the problem as a standard *catastrophic forgetting issue*, a phenomenon which can be observed as a dramatic performance degradation when some new tasks are added to an existing NN model. To tackle this fundamental problem [37], lifelong learning is one of the solutions which aims to preserve the performance on previous tasks while adapting to new data. Hence, we employ lifelong learning on the proposed ABR system for continuously training the NN to fit the dynamic changes of networks. Furthermore, in order to further reduce the training overhead, we implement a module that can dynamically filter the useful network traces from the traces which instantly collected from the clients.

IV. Comyco System Overview

In this section, taking the above challenges into account, we propose *Comyco*, an ABR system that uses the lifelong imitation learning method to update the NN continuously. In detail, as illustrated in Figure 7, Comyco consists of two sub-systems, i.e., *inner-loop training system* and *outer-loop system*. The inner-loop system adopts the imitation learning method to efficiently learn the policy via *cloning* the behavior of the expert strategy. The outer-loop system enables Comyco to keep updating with low extra overhead. It's notable that the inner-loop system can be deployed solely if there is no need to continuously update the model. The Comyco's system workflow is shown as follows: before the video starts, the video player, placed on the client-side, downloads the latest NN model from *ABR model server* for making further decisions. Once the video session ends, the player collects the available throughput trace via past download chunk size and download time. At the same time, the collected trace will be submitted to the outer-loop system, which is placed on the server-side. Then the outer-loop system will instantly compute the gap between the current policy and the optimal strategy of the submitted trace and determine whether the trace should be *learned* by current NN during the next training loop. Next, for each time duration, the inner-loop training system, also placed on the server, will be enabled by the outer-loop system. It then updates the NN w.r.t the selected traces efficiently via lifelong imitation learning training method. Finally, the trained model will be frozen and submitted to the ABR model server. The server then starts waiting for the request of newcomer players.

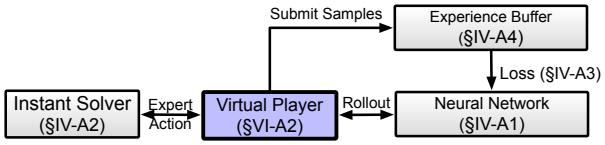


Fig. 8. Inner-loop Training System Work-flow Overview. Training methodologies are available in §IV-A4.

A. Inner-loop System Overview

Comyco's inner-loop system work-flow is illustrated in Figure 8. The sub-system is mainly composed of a NN, an ABR virtual player, an instant solver, and an experience replay buffer. We start by introducing Comyco's NN architecture. Then we explain the basic training methodology. Finally, we further illustrate Comyco with a multi-agent framework.

1) NN Architecture Overview: Motivated by the recent success of no-regret online learning methods [38], Comyco's learning agent is allowed to explore the environment via traditional rollout methods. For each epoch t, the agent aims to select the next bitrate via a NN. We now explain the details of the agent's NN including its inputs, outputs, network architecture, and representation.

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 $C_k = \{c_{k-t-1}, \ldots, c_k\}$ 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 which stands for the perceptual video quality metrics for each bitrate of the next chunk.

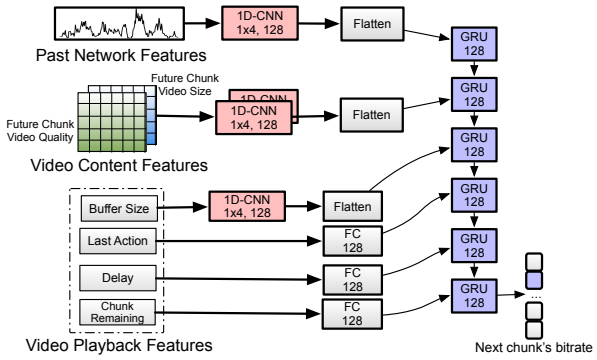


Fig. 9. Comyco's NN architecture Overview. The NN contains network features, video content information, as well as playback status.

▷ ***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.

NN Representation. As shown in Figure 9, for each input type, we use a proper and specific method to extract the underlying features. Specifically, 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 a 1D-CNN or a fully connected layer to extract the useful characteristics from each metric upon the video playback inputs. The selected features are passed into a Gated Recurrent Unit (GRU) [39] 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 use *RelU* as the active function for each feature extraction layer and leverage *softmax* for the last layer.

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 the *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 [10], we formulate the problem as demonstrated in Eq. 2, denoted as QoE^N_{max}. In detail, the virtual player consists of a virtual timestamp, 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 [4]. 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 §VI-A5. In this work, we set $N = 8$.

$$\max_{R_1,\dots,R_N,T_s} QoE^N \tag{2}$$

$$\text{s.t.} \quad t_{k+1} = t_k + \frac{d_k(R_k)}{C_k} + \delta t_k, \tag{3}$$

$$C_k = \frac{1}{t_{k+1} - t_k - \delta t_k} \int_{t_k}^{t_{k+1}-\delta t_k} C_t dt, \tag{4}$$

$$B_{k+1} = \left[\left(B_k - \frac{d_k(R_k)}{C_k} \right)_+ + L - \delta t_k \right]_+, \tag{5}$$

$$B_1 = T_s, \tag{6}$$

$$B_k \in [0, B_{max}], R_k \in R, \forall k = 1, 2, \dots, N. \tag{7}$$

3) Choice of Loss Functions for Comyco: We start by 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 $q_\pi(s, a)$ [35]. Thus, given an expert action $\hat{a}$ and the optimal value function $q_\pi(s, \hat{a}) = q_*(s, a)$, we can update the model via minimizing the gap between the true action probability $\hat{A}$ and π, where $\hat{A}$ is a one hot encoding in terms of $\hat{a}$. For more theoretical analysis please refer to §VII-A. In this paper, we use cross entropy error as the loss function. Note that the function can be represented as any traditional behavioral cloning loss method [22], 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 [40], we further add the entropy H 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 8.

$$L_{comyco} = -\sum \hat{A} \log \pi(s, a; \theta) + \alpha H(\pi(s; \theta)). \tag{8}$$

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 hyper-parameter that controls the encouragement of exploration. In this paper, we set $\alpha = 10^{-3}$ and discuss L_{comyco} with different α in §VI-A5. Recall that $L_{lifelong}$ (§IV-B3) will be used to take the place of L_{comyco} if the outer-loop system is required.

4) Training Comyco with Experience Replay: Recent off-policy RL-based methods [41] leverage experience replay buffer to achieve better convergence behavior when training

a function approximator. Inspired by the success of these approaches, we also create a sample buffer that can store the past expert strategies and allow the algorithm to randomly pick the sample from the buffer during the training process. We will discuss the effect of utilizing experience replay in §VI-A5. We summarize the training procedure in Alg. 1.

Algorithm 1 Inner-loop Overall Training Procedure

Require: Training model θ, `Instant Solver`(§IV-A2).
procedure INNER-LOOP TRAINING
  Initialize π.
  Sample Training Batch $B = \{\}$.
  Randomly pick *trace*, *video* from network (§VI-A2) and video (§V-B) dataset.
  Get State ABR state s_t.
  repeat
    Picks a_t according to policy $\pi(s_t; \theta)$.
    Expert action $\hat{a}_t$ = `Instant Solver`$(s_t, trace, video)$.
    $B \leftarrow B \bigcup \{s_t, \hat{a}_t\}$.
    Samples a batch $\hat{B} \in B$.
    Updates network θ with $\hat{B}$ using Eq.8 or Eq.10;
    Produces next ABR state S_{t+1} according to s_t and a_t.
    if done **then** ▷ End of the video.
      Randomly pick *trace*, *video* from the network and video dataset.
      Get State ABR state s_t.
    $t \leftarrow t + 1$
  until Converged

5) Parallel Training: Notably, the training process can be designed asynchronously, which is quite suitable for the multi-agent parallel training framework. Inspired by the multi-agent training method [18], [40], we modify Comyco's framework from single-agent training to asynchronous multi-agent training. The 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 the 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. By default, Comyco uses **12** agents, which is the same number of CPU cores of our PC.

B. Outer-loop System Overview

The key idea of the outer-loop sub-system is to reduce the number of training set with guaranteeing the training performance as much as possible. We demonstrate the outer-loop lifelong learning system in Figure 10. The sub-system includes several modules, such as the Optimal Estimator (§IV-B1) and the Trace Collector (§IV-B2). In this section, we introduce the modules and illustrate the training methodologies (§IV-B3).

1) Optimal Estimator: Technically, the Optimal Estimator is a module which can compute the normalized QoE E_{tr} on a network throughput trace tr, where the trace is collected and reported from the client. As suggested by prior work, E_{tr} is defined as the ratio between the QoE performed by the current policy $Q_{tr,\theta}$ and the optimal QoE Q_{opt} (Eq. 9). Meanwhile,

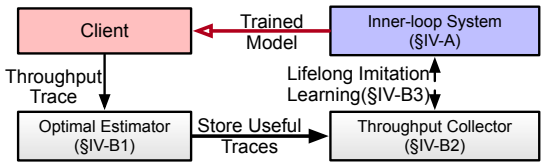


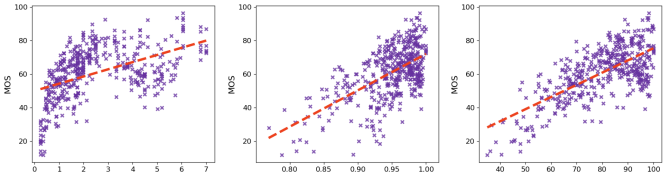
Fig. 10. Outer-loop Training System Work-flow Overview. Training methodologies are available in §IV-B3.

we leverage an instant solver (§IV-A2) for estimating the optimal strategy. Specifically, we roll out the best bitrate for each step via maximizing the QoE objective (Eq. 2). In this work, we also set the future horizon $N = 8$ since the near-optimal policy is well enough for this task [10], [12]. It's worth noting that we can also employ Hindsight [4] or another optimal ABR estimator [5] to replace the instant solver instead.

$$E_{tr} = \frac{Q_{tr,\theta}}{Q_{opt}} \tag{9}$$

2) Trace Collector: Having computed the normalized QoE metric from the Optimal Estimator, we focus on learning a proper NN for the current network conditions efficiently. Ideally, the intuitive idea is to use the whole collected trace to fine-tune the old NN. However, it's impractical since the number of collected trace is so huge that the NN can not be trained in an allowable time. Moreover, previous analysis shows that there is not much difference in the network distribution within an hour (§III-B), that means, the trained policy may perform well on most traces but eventually fail on some traces. Taking such observations into account, our key idea is to pick proper traces into the Trace Collector, in which the normalized QoE of the trace is lower than the given threshold `Thres`. The inner-loop system then trains the NN from the network trace in the Trace Collector. Finally, Comyco generalizes a strategy for the network conditions under the next training period. In this work, we set `Thres` as 0.8, and the training period as 1 hour. We further investigate the influence of `Thres` on the proposed method in §VI-B7.

3) Loss Function for Lifelong Learning Method: The goal of the lifelong learning [42] is to avoid Catastrophic Forgetting, that means, the NN works well on the latest task but suffers from unexpected performance on previous tasks. In this work, we pick Learning without Forgetting (LwF) [23], which uses outputs of the old models as soft targets on old tasks, as the learning algorithm. LwF enables the lowest computation cost among all the previously proposed schemes and works in the comparable performance in terms of the state-of-the-art approach [42]. Subsequently, we implement an LwF-based loss function $L_{lifelong}$ to take the place of the loss function L_{Comyco} of the inner-loop system for achieving continual learning. The equation is listed in Eq. 10, where L_{Comyco} represents the loss function of Comyco (listed in Eq. 8), $\pi(s, a; \theta_{old})$ is roll out policy via the old NN (previous network), and $\pi(s, a; \theta)$ is the rollout policy for current NN. It is notable that we refer the old policy $\pi(s, a; \theta_{old})$ as a value, which means, it does not provide any gradients for the loss function. Inspired by prior work [23], we set λ=1. In general,



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

Fig. 11. Correlation comparison of video presentation quality metrics on the SQoE-III dataset [21]. Results are summarized by Pearson correlation coefficient [43].

we demonstrate the training methodology of the outer-loop system in Alg. 2. As shown, the workflow mainly consists of three parts, i.e., i) picking necessary network traces via Optimal Estimator, ii) storing the trace if the normalized QoE lower than the threshold, iii) starting learning the algorithm if the period reaches 1 hour.

$$L_{lifelong} = L_{Comyco} + \lambda \sum \pi(s;\theta_{old}) \log \pi(s;\theta). \quad (10)$$

Algorithm 2 Outer-loop Overall Training Procedure

Require: Old model θ_{old}, Training model θ
Require: *Trace Collector* T (§IV-B3), Threshold `Thres`.
Require: *Optimal Estimator*(§IV-B1).
1: **procedure** OUTER-LOOP TRAINING
2: Receive *trace* from the client.
3: $E_{tr} \leftarrow$ *Optimal Estimator*(*trace*).
4: **if** E_{tr} <`Thres` **then**
5: Store *trace* into *Trace Collector*.
6: **if** Training period $=$ *1 hour* **then**
7: Inner-loop Training(T) (§IV-A4) with the loss function $L_{Lifelong}$ (Alg.10).
8: Submit the trained model to the ABR model server.

V. QoE Metrics and Video Datasets

Upon constructing Comyco's NN architecture by 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 the real-world ABR QoE dataset [21]. Finally, we collect a DASH video dataset with different VMAF assessments.

A. QoE Model Setup

Motivated by the linear-based QoE metric that is widely used to evaluate several ABR schemes [10], [12], [17], [28], [32], [44], we concluded our QoE metric $\texttt{QoE}_v$ as:

$$\texttt{QoE}_v = \alpha \sum_{n=1}^{N} q(R_n) - \beta \sum_{n=1}^{N} T_n + \gamma \sum_{n=1}^{N-1} [q(R_{n+1}) - q(R_n)]_+ - \delta \sum_{n=1}^{N-1} [q(R_{n+1}) - q(R_n)]_-, \quad (11)$$

TABLE I
PERFOMANCE COMPARISON OF QOE MODELS ON WATERLOO STREAMING SQOE-III [21]

QoE model	Type	VQA	SRCC
Pensieve's [12]	linear	-	0.6256
MPC's [10]	linear	-	0.7143
Bentaleb's [28]	linear	SSIMplus [48]	0.6322
Duanmu's [21]	linear	-	0.7743
QoE_v with Combined Smooth.	linear	VMAF [20]	0.7741
Comyco's	linear	VMAF	**0.7870**

where N is the total number of chunks during the session, R_n represents 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, and α, β, γ, δ are the parameters to describe their aggressiveness.

Choice of $q(R_n)$. Estimating QoE via handcrafted features from the client-side has lasted a long history [33], as several schemes seldom yield a reliable result. Revisiting these schemes, we find that Video quality assessment (VQA) plays a crucial part in QoE models. Most studies pick video bitrate, SSIM or PSNR [45] as the inputs, while such metrics fail to either precisely reflect the visual quality seen by human eyes or accurately describe the latent video features, resulting in the failure of characterizing the video qualities of the entire video sessions [31]. 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 and Video Multimethod Assessment Fusion (VMAF) [20], under the Waterloo Streaming QoE Database III (SQoE-III). Here SQoE-III is the *largest and most realistic dataset* for dynamic adaptive streaming over HTTP [21], which consists of a total of 450 streaming videos created from diverse source content and diverse distortion patterns [21]. SSIM is a popular image quality metric [13]. VMAF is an objective full-reference video quality metric that is formulated by Netflix to estimate subjective video quality. Results are collected with Pearson correlation coefficient [43] as suggested by [46]. As shown in Figure 11, we can see that VMAF achieves the highest correlation among all candidates, with the improvements in the coefficient of 16.39%-43.54%. Besides, VMAF is also a popular scheme with great potential in both academia and industry [47]. We, therefore, set $q(R_n) = \texttt{VMAF}(R_n)$.

QoE Parameters Setup. Recall that the 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 [21] 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 take the Spearman correlation coefficient (SRCC), as suggested by [21], 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 I. As shown, the QoE_v model outperforms recent work. In conclusion, the proposed QoE model is well enough to evaluate ABR schemes.

Separated smoothness metrics. The reason why we separate smoothness metrics is that: during the pre-experiment, we find that there is a positive correlation between positive smoothness and MOS, which means, users will feel satisfied if the video quality increases. Extensive analysis shows that the weight for negative smoothness is 3× higher than that of positive smoothness, which demystifies a severe penalty on decreasing video qualities. Besides, prior work [33], [49], [50] has also observed the correlation between the positive smoothness and the negative smoothness. Results on Table I also illustrate that Comyco with separated smoothness metric can effectively improve the performance on the SRCC score of 3.0%.

B. Video Datasets

To better improve the Comyco's generalization ability, we propose a video quality DASH dataset that involves movies, sports, TV-shows, games, news and MVs. Specifically, we first collect video clips with highest resolution from YouTube [51], then use FFmpeg [52] to encode the video by H.264 codec and MP4Box [53] to *dashify* videos according to the encoding ladder of video sequences ({235, 375, 560, 750, 1050, 1750, 2350, 3000, 4300}kbps) [12], [21], [25]. 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. The dataset have been published in [54].

VI. Evaluation

In this section, we propose several experiments to analyze the performance of Comyco. We start by evaluating Comyco's inner-loop system under various network conditions and compare it with previously proposed ABR approaches (§VI-A). We then evaluate the outer-loop system over real-world network traces and compare it with several ABR schemes, such as previously proposed ABR schemes, and outer-loop system with different updating strategies (§VI-B).

A. Evaluation for Inner-loop System

Recall that the Comyco's inner-loop system can be deployed solely if there is no need to achieve continual learning. In this experiment, we treat the inner-loop system as *Comyco*, and use L_{comyco} as the NN's loss function.

1) Implementation: We use TensorFlow [55] to implement the training workflow and utilizing TFlearn [56] to construct the NN architecture. Besides, we use C++ to implement the instant solver and the virtual player. Then we leverage Swig [57] to compile them as a Python class. The NN takes the past sequence length $k = 8$ (as suggested by [12]) and future 1 video chunk features (as suggested by [10]) into the NN. We set the learning rate $\alpha = 10^{-4}$ and use the Adam optimizer [58] to optimize the model. For more details, please refer to our repository [59].

2) Experimental Setup: The evaluation system consists of:

Virtual Player. We design a faithful ABR offline virtual player to train Comyco via network traces and video descriptions. The player is written in C++ and Python3.6, with close refering to several state-of-the-art open-sourced ABR simulators including Pensieve, Oboe and Sabre [11]. Comparing the executing time of C++-based instant solver and python-based solver, we find that using C++ will significantly accelerate the training process, with the improvements of 15,000%.

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. The testbed is composed of:

- **Trace-driven Emulation.** Following the instructions of recent work [12], [17], we utilize Mahimahi [60] 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 §VI-A6.

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

- **Chunk-level Network Traces:** including HSDPA [27]: 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 [26]: a broadband dataset (1000 traces, 1s granularity); Oboe [61] (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 [12]. 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. Details of each algorithm are listed in §VIII.

- **Rate-based Approach (RB) [7]:** uses harmonic mean of past five throughputs measured as future bandwidth, and picks the next chunks' bitrate with nearest and lower than the predicted bandwidth.
- **BOLA [9]:** turns the ABR problem into a utility maximization problem and solve it by using the Lyapunov function. It is a typical buffer-based approach. We use BOLA provided by the authors [11].
- **RobustMPC [10]:** 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 (§V-A) to optimize the strategy.
- **Pensieve [12]:** the state-of-the-art ABR scheme which utilizes Deep Reinforcement Learning (DRL) to pick bitrate for next video chunks. Pensieve takes the former network

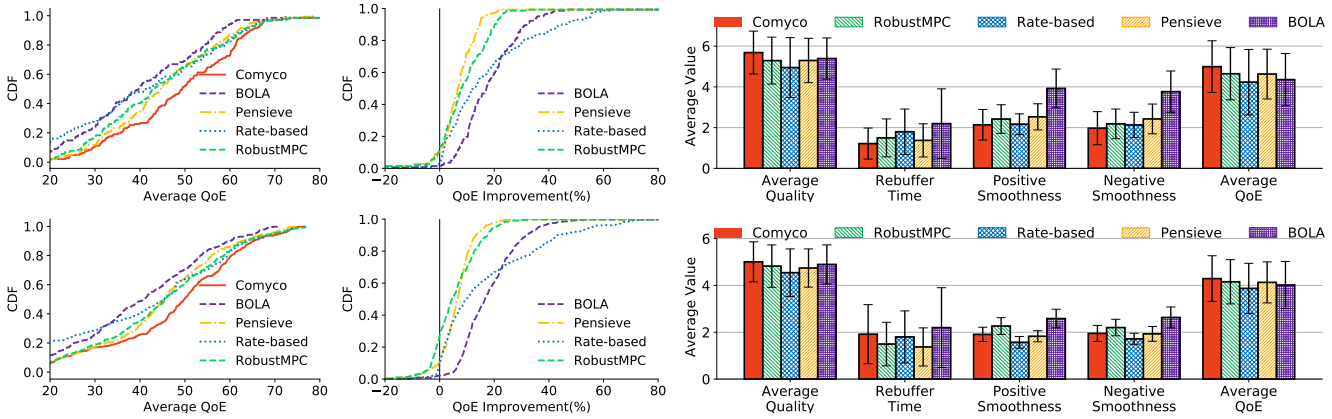
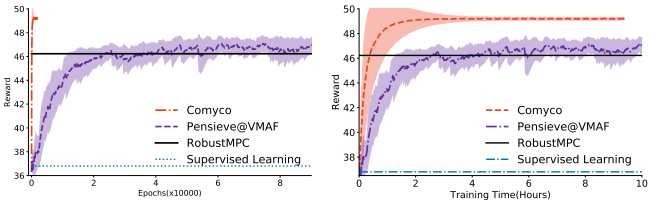


Fig. 12. Comparing Comyco with existing ABR approaches under the HSDPA and FCC network traces. Results are illustrated with CDF distributions, QoE improvement curves and the comparison of several underlying metrics (§V-A).



(a) Epochs (b) Training Time

Fig. 13. 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.

status as states and reinforces itself through the interaction with the faithful offline simulator. We use the scheme implemented by the authors [30] but retrain the model for our work (§VI-A3).

3) 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 §VI-A2. We use *EnvivoDash3*, a widely used [10], [12], [17], [44] reference video clip [25] and QoE_v to measure the ABR performance.

▷ ***Pensieve Re-training.*** We retrain Pensieve via our datasets (§VI-A2), NN architectures (§IV-A1) and QoE metrics (§V-A). Followed by recent work [17], our experiments use different entropy weights in the range of 5.0 to 0.1 and dynamically decrease the weight every 1000 iterations. The 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.

Comyco vs. Existing ABRs. Figure 12 shows the comparison of QoE metrics for existing ABR schemes (§VI-A2). 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 improvements 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 performed on the average quality metric, which improves 6.84%-15.64% compared with other ABRs. Moreover, Comyco is able to avoid rebuffering and bitrate changes.

Sample Efficiency of ABR Schemes. Figure 13 illustrates the average QoE of learning-based ABR schemes under the HSDPA network traces. We validate the performance of two schemes respectively during the training process. Results are shown with two perspectives including Epoch-Average QoE and Training time-Average QoE. As expected (§III-B), we observe that the supervised learning-based method fails to find a strategy, which thereby leads to poor performance. Furthermore, we see about **1700x** improvement in terms of the number of samples required and about **16x** improvement in terms of training time required. It makes sense since the training the agent with a model-free RL-based method [35] is difficult. The agent is required to learn a latent representation together with a control policy to perform the task, as generalizing a feasible encoder via a continuous reward signal is not only extremely sample inefficient but also prone to suboptimal convergence. Meanwhile, achieving high sample efficiency is essential since the equilibria of learning-based methods are not always Pareto efficient in offline-training tasks, e.g., fast generating ABR algorithms for personalized QoE or specific videos, new NN architecture exploration. Moreover, the key issue of RL-based ABR scheme (e.g., Pensieve) is to neglect exogenous inputs (e.g. future throughput measured) when estimating policy gradient, since the advantage function may be overestimated or underestimated. On the contrary, in our work, Comyco considers the future throughput as the latent feature. As a result, imitation learning-based ABR approach Comyco outperforms Rl-based ABRs. Same conclusions and proofs please refer to RL-based input variance algorithms [34].

4) Comyco with Multiple Videos: To better understand how Comyco performs on various videos, we randomly pick videos from different video types (§V-B) and utilize Oboe network traces [17] to evaluate the QoE_v performances of the proposed methods. Oboe network traces have diverse

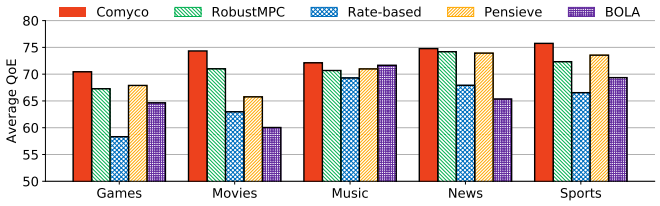


Fig. 14. Comparing Comyco with existing ABR approaches under the Oboe network traces and various types of videos.

TABLE II
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

network conditions, which bring more challenges for us to improve the performance. Figure 14 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 is really an interesting topic and we will discuss it in future work.

5) Ablation Study: In this section, we set up several experiments that aim to provide a thorough understanding of Comyco, including its hyper-parameters and overhead. It is worth noting that we have computed the offline-optimal results via dynamic programming and complete network status [12] 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 experience strategy in Table II. Results are collected under the Oboe dataset [17]. As shown, we find that experience replay can effectively help Comyco learn better. Recall that the instant solver is only used in the training process, and Comyco will inference solely on the client side during the validation process. Meanwhile, despite the outstanding performance of Comyco with N=9, such scheme lacks the algorithmic efficiency and can hardly be deployed in practice. Thus, we choose N=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.

TABLE III
COMYCO WITH DIFFERENT α.

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

TABLE IV
MODEL SIZE AND COST COMPARISON OF DIFFERENT ABRS.

	RB [7]	BB [8]	Quetra [62]	RMPC [10]	Pensieve [12]	*Comyco*
Size(MB)	**0.003**	0.003	0.005	0.013	2.6	2.4
Time(MS)	461	**278**	588	10854	3090	2593

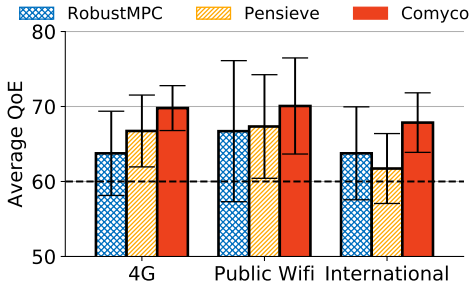


Fig. 15. Comparing Comyco with Pensieve and RobustMPC under the real-world network conditions. We take $QoE = 60$ as baselines.

As listed in Table III, we confirm that $\alpha = 0.001$ represents the best parameters for our work. Meanwhile, results also prove the effectiveness of utilizing entropy loss (§IV-A3).

Comyco Overhead. We calculate [63] 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 [64] (146 Mflops). At the same time, we also discuss the model size and time span of several representative ABR algorithms in Table IV, in which the time span represents the total time taken by the algorithm to execute about 7,000 times. The experiment is done on the 12-core, Intel i7 CPUs with 32GB RAM. As shown, the average execution time of Comyco on the laptop is only 0.4ms, yielding an acceptable result. Hence, we believe that Comyco can be successfully deployed on the PC and laptop, or even, on the mobile.

6) 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` [25] 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 (from Beijing to Qingdao), WiFi network (from Tsinghua's public WiFi to Qingdao) and inter-

TABLE V
REAL-WORLD NETWORK MEASUREMENT.

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

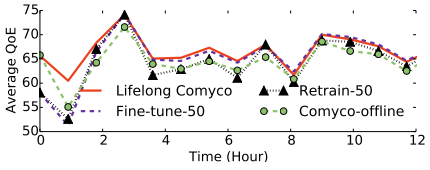


Fig. 16. Comparing Comyco with several baselines under Kwai dataset. Results are reported with QoE curves on each duration.

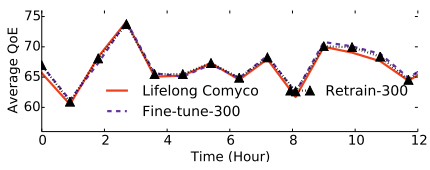


Fig. 17. Comparison of Comyco and online-optimal under Kwai dataset. Results are reported with QoE curves on each duration.

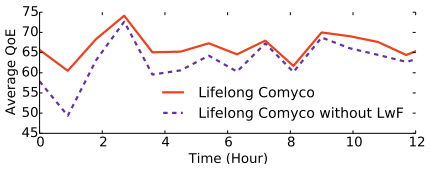


Fig. 18. Comparing the QoE of Comyco with the one without using LwF method. Results are collected under the Kwai dataset.

national link (from Singapore to Beijing). Table V illustrates network status, where μ is the average throughput measured and σ represents standard deviation from the average. For each round, we randomly pick a scheme from candidates and summarize the bitrate selected and rebuffering time for each chunk. Each experiment takes about 2 hours. Figure 15 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.

B. Evaluation for Outer-loop System

1) Implementation: We adopt C++ to implement the Optimal Estimator, and uses Python to construct the Trace Collector. Note that the inner-loop system uses $L_{lifelong}$ (§10), rather than L_{comyco} (§8), to train the NN, since the outer-loop system enables Comyco to achieve continual learning.

2) Experimental Setup: Considering the goal is to evaluate the effectiveness of lifelong learning rather than the performance of ABR streaming, we adopt virtual player (§VI-A2 to validate the outer-loop system via trace-driven emulation. Technically, unlike inner-loop system evaluation, we list network trace dataset and baselines as follows:

Network Trace Dataset. As described before, the sub-system is required to evaluate on continuous network throughput dataset. To that end, we utilize the large-scale network bandwidth dataset Kwai. The dataset contains over 860,000 traces, collected from about 10,000 unique users, totally 7 days from various network conditions, including wired, WiFi, cellular network, and so forth (§III-B).

Baselines. In this work, we pick several representative outer-loop system in different strategies as baselines.

- ▷ **Fine-tuning for 50 epochs (*Fine-tune-50*):** for each period t, we tune the trained model on the t-th hours' network traces for 50 epochs, lasting about 5 minutes for learning.
- ▷ **Fine-tuning for 300 epochs (*Fine-tune-300*):** we fine-tune the trained model on network traces for 300 epochs for each time period. Note that the training time lasts over 30 minutes, which is impractical in practice.
- ▷ **Re-train Comyco in 50 epochs (*Retrain-50*):** for each duration t, we train Comyco on network traces in the range of t-th hours from scratch.
- ▷ **Re-train Comyco in 300 epochs (*Retrain-300*):** we retrain Comyco for about 300 epochs since Comyco will be efficiently converged with an acceptable results. Recall that the training time lasts over half of the time duration.
- ▷ **RobustMPC [10]:** picks the bitrate by the model predictive control method. As mentioned before, we also adopt C++ to implement *RobustMPC* and leverage QoE_v (§V-A) to optimize the strategy.
- ▷ **Comyco Offline Training (*Comyco-offline*):** offline trains Comyco with inner-loop system's network trace dataset (§VI-A2). Note that we didn't further tune the NN model once Comyco's model has been trained (§VI-A5).

Testing Methodology. Like previous experiments, we use QoE_v (§V-A) to evaluate each scheme. For each duration t, we train the baselines on the network throughput traces with the range of t-th to t+1-th hour in the Kwai dataset, and validate them on the network dataset within t+1-th to t+2-th hour. In order to evaluate the fast convergence, we set the default training epoch as 50. The training time lasts about 5 minutes on our device (§VI-A2), and we evaluate the traces of 12 hours in one day. Furthermore, we also set *training models in 300 epochs (Fine-tine-300 and Retrain-300)* as strong baselines to better understand the gap between the proposed strategies and online-optimal policies. In this experiment, we treat the Comyco with outer-loop system as *lifelong Comyco*.

3) Comparison of Different Outer-loop Strategies: Figure 16 shows the average QoE curves of lifelong Comyco and other baselines on the Kwai dataset. We can see that lifelong Comyco always rivals or outperforms other approaches. Specifically, lifelong Comyco performs better than the Comyco-offline scheme, with the improvements on average QoE of 1.07% - 9.81%. Another observation of this experiment demonstrates the weakness of the retrain and the fine-tune approach: if the network situation changes dramatically (see time 0 to time 2 in Figure 16), such algorithms will not be able to provide reliable QoE to the users. Besides, fine-tune-50 works well when the network distribution changes steadily (from time 4-7) since the training set and the validation set are almost in the same distribution.

4) lifelong Comyco vs. Comyco without LwF: In this experiment, we validate the effectiveness of the LwF method. As shown in Figure 18, comparing the overall QoE performance of lifelong Comyco and Comyco without LwF, we observe that lifelong Comyco improves the average QoE by 1.51%-21.41% compared with the other methods. It makes sense since lifelong Comyco trains the NN with joint considering the previous network status and current network observed, whereas the other one diverges.

5) lifelong Comyco vs. Online Optimal: To better understand the gap between lifelong Comyco and online optimal, we set up an experiment to evaluate the performance of lifelong Comyco, Fine-tune-300 as well as Retrain-300 (§VI-B2).

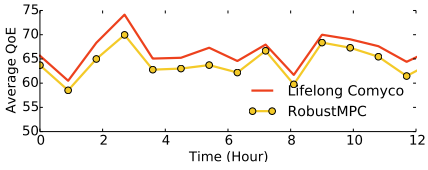


Fig. 19. Comparison of Comyco and state-of-the-art model-based algorithm RobustMPC under Kwai dataset. Results are also reported with QoE curves on each duration.

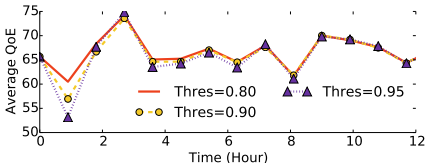


Fig. 20. Comparing the QoE of Comyco with the one without using LwF method. Results are collected under the Kwai dataset.

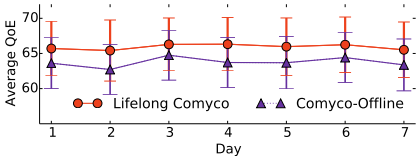


Fig. 21. Comparing the performance of Comyco and Comyco-Offline performs over multiple days.

Results are collected over the same video description and network traces. As illustrated in Figure 17, we show that lifelong Comyco almost reaches the online optimal across the entire session, with the decreases of only 0.02%-3.34% compared with Fine-tune-300, and 0.12%-3.33% in terms of Retrain-300. In particular, we also find that lifelong Comyco performs better than 16% of the sessions on Fine-tune-300 and Retrain-300, where the QoE performance slightly increases with the range of 0.17% to 1.53%. Such a conclusion also proves the effectiveness of the lifelong learning method.

6) lifelong Comyco vs. RobustMPC: Besides, we also compare the performance of lifelong Comyco with the current state-of-the-art model-based approach RobustMPC. Results are illustrated as QoE curves in Figure 19. As expected, we can find that lifelong Comyco stands for the better scheme, outperforming RobustMPC on average QoE of 0.12% - 5.70%. In general, such observations prove that a good ABR algorithm is required to update dynamically for fitting the changes of real-world network conditions [17].

7) lifelong Comyco with Different Threshold `Thres`*:* In this experiment, we aim to understand the influence of threshold `Thres` for Comyco. In detail, we use three threshold candidates, involving $\{0.8, 0.9, 0.95\}$. We evaluate the lifelong Comyco with the proposed threshold on the same network environments respectively. Results are plotted in Figure 20. As shown, we see that `Thres`=0.8 represents the best parameter of lifelong Comyco. Especially, `Thres`=0.8 works well in time 2, while the other scheme fails to achieve a good result. It is notable that the choice of the value strongly depends on the current task.

8) Evaluating lifelong Comyco Throughout the Entire Session: Finally, we discuss the behavior of lifelong Comyco and Comyco-Offline over multiple days. Recall that once the Comyco-Offline has been trained, the model is not allowed to be fine-tuned with any methods (we can also call this zero-shot learning). Result in Figure 21 illustrates that the lifelong Comyco can always keep the performance within a stable range. In contrast, Comyco-Offline sometimes fails to perform well on some days (e.g., the 4-th day) since it cannot adapt to time-vary network environments. In general, lifelong Comyco improves the average QoE by 2.3%-4.2% compared with Comyco-Offline. Note that this is rather not a minor improvement because it is difficult to improve the average performance of the huge dataset. For example, CS2P [5] increased the QoE by 3.2% compared with MPC [10], and ABRL [65] improved the video quality by 1.6% compared with Pensieve.

VII. Discussion

A. Theoretical Analysis

In this work, the Comyco's inner-loop method (§IV-A4) can be defined as a no-regret algorithm because it produces a sequence of policies $\pi_1, \pi_2, \ldots, \pi_N$ such that the average regret w.r.t the best policy in hindsight goes to 0 as N goes to ∞: $\frac{1}{N}\sum_{i=1}^{N}\ell_i(\pi_i) - \min_{\pi\in\Pi}\frac{1}{N}\sum_{i=1}^{N}\ell_i(\pi) \leq \gamma_N$, for $\lim_{N\to\infty}\gamma_N = 0$. Here ℓ_n represents any strongly convex surrogate loss loss functions, such as mean square error and cross entropy error. The loss function ℓ_n is allowed to be optimized by any optimization algorithm (e.g., Adam). Thus, inspired by the prior work [38], let $\hat{\pi}_i$ denote the policy that minimizes the observed loss, we have to bound the total variation distance between the distribution of states encountered by $\hat{\pi}_i$ and π_i as follows:

Lemma VII.1. $||d_{\pi_i} - d_{\hat{\pi}_i}||_1 \leq 2\beta_i T$.

Proof. Let $d\pi$ denote the average distribution of states if we follow policy π for T steps, d reflects the distribution of states over T steps conditioned on π_i picking expert's policy π^* at least once over T steps, β represents the probability of π_i selecting π^*. We have:

$$\begin{aligned}
&||d_{\pi_i} - d_{\hat{\pi}_i}||_1 \\
=\ & ||(1-\beta_i)^T d_{\hat{\pi}_i} + (1-(1-\beta_i)^T)d - d_{\hat{\pi}_i}||_1 \\
=\ & [1-(1-\beta_i)^T]||d - d_{\hat{\pi}_i}||_1 \\
\leq\ & 2[1-(1-\beta_i)^T] \\
\leq\ & 2[1-(1-\beta_i T)] \\
\leq\ & 2\beta_i T.
\end{aligned}$$

Notice that in Comyco, β_i is NOT a fixed value and is strongly correlated with the entropy $H(\cdot)$ of the policy π_θ: $\beta_i \propto H(\cdot)$. Hence, it's critical to set the proper entropy weight α in the Comyco's loss function (§VI-A5). □

Let $\epsilon_N = \min_{\pi\in\Pi}\frac{1}{N}\sum_{i=1}^{N}\mathbb{E}_{s\sim d_{\pi_i}}[\ell(s,\pi)]$ the loss of the best policy in hindsight after N iterations and let $\ell_{\max}$ be an upper bound on the loss and state s s.t. $d_{\hat{\pi}_i}(s) > 0$. We have:

Theorem VII.1. *For Comyco, there exists a policy* $\hat{\pi} \in \hat{\pi}_{1:N}$ *s.t.* $\mathbb{E}_{s\sim d_{\hat{\pi}}}[\ell(s,\hat{\pi})] \leq \epsilon_N + \gamma_N + \frac{2\ell_{\max}}{N}[n_\beta + T\sum_{i=n_\beta+1}^{N}\beta_i]$, *for* γ_N *the average regret of* $\hat{\pi}_{1:N}$.

Proof. As mentioned before, Lemma VII.1 implies that $\mathbb{E}_{s\sim d_{\hat{\pi}_i}}[\ell_i(s,\hat{\pi}_i)] \leq \mathbb{E}_{s\sim d_{\pi_i}}[\ell_i(s,\hat{\pi}_i)] + 2\ell_{\max}\min(1,\beta_i T)$.

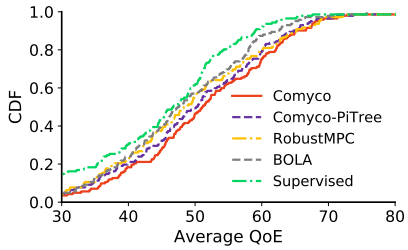


Fig. 22. Comparing the QoE of Comyco-Pitree with the trained Comyco and RobustMPC. Results are collected under the HSDPA dataset.

$$
\begin{aligned}
&\min_{\hat{\pi}\in\hat{\pi}_{1:N}} \mathbb{E}_{s\sim d_{\hat{\pi}}}[\ell(s,\hat{\pi})] \\
\leq\ & \tfrac{1}{N}\textstyle\sum_{i=1}^{N} \mathbb{E}_{s\sim d_{\hat{\pi}_i}}(\ell(s,\hat{\pi}_i)) \\
\leq\ & \tfrac{1}{N}\textstyle\sum_{i=1}^{N}[\mathbb{E}_{s\sim d_{\pi_i}}(\ell(s,\hat{\pi}_i)) + 2\ell_{\max}\min(1,\beta_i T)] \\
\leq\ & \gamma_N + \tfrac{2\ell_{\max}}{N}[n_\beta + \textstyle\sum_{i=n_\beta+1}^{N}\beta_i T] + \min_{\pi\in\Pi}\textstyle\sum_{i=1}^{N}\ell_i(\pi) \\
=\ & \gamma_N + \epsilon_N + \tfrac{2\ell_{\max}}{N}[n_\beta + \textstyle\sum_{i=n_\beta+1}^{N}\beta_i T]
\end{aligned}
$$

Under an error reduction assumption that for any input distribution, there is some policy $\pi \in \Pi$ that achieves surrogate loss of ϵ, which implies we are guaranteed to find a policy $\hat{\pi}$ that achieves ϵ under $H(\cdot) \to 0$. In Comyco, the policy's entropy is allowed to decrease effectively via the cross entropy method. Moreover, many no-regret algorithms (e.g., DAgger [38]) guarantee that if β_i is chosen to be the form of $(1-\alpha)^{i-1}$, in which α is a constant hyper-parameter, then the method need at least $\tilde{O}(T)$ iterations to make γ_N negligible. □

B. Practical Implementation

Learning-based ABR algorithms are struggling with its deployability. Specifically, Pensieve [12] is deployed on the server to avoid high computational costs on the client-side. However, in practice, most ABR algorithms are executed in the front-end to avert the extra latency connecting to the back-end [66], [67]. Thus, such ABR policy frameworks ([12], [17]) are theoretically effective but impractical [65]. In this paper, as much as this work is NOT focused on the deployable problem of learning-based ABR algorithms, we still give some practical ideas for implementation.

- We argue that deploying the model on the client is impractical since the computational cost is rather small for today's mobile (§VI-A5). What's more, the user will optionally download the small-sized model, where the model size is even 50% smaller than the lowest video chunk size (by Tensorflow.js [68]).
- Several practical ABR schemes (i.e., PiTree [69], LIME [70] and ABRL [65]) have been proposed to distill the NN to a practical decision tree, an interpretable tabular, or a linear-based formula. Such schemes are also acceptable for appending into the Comyco system. For example, we use PiTree [69] to distill the trained Comyco to a decision tree model and show the CDF results on Figure 22, where the results are collected under the HSDPA dataset. We can see that Comyco-Pitree decreases the model size of about 97% with preserving the overall performance.
- Recent years have also seen several schemes that deploy the ABR algorithm as a service on the cloud. For example, Sun et al. [5] assume that throughput factors can be efficiently captured by Hidden-Markov-Model (HMM), then they optimize the model on the cloud with huge amounts of data. Oboe [17] attempts to place a dictionary, mapping the throughput status {average throughput μ, throughput variance σ} to the optimized traditional ABRs' ([9], [10]) parameters, on the cloud for assisting traditional algorithms to achieve higher performances in different network conditions. Meanwhile, deploying ABRs on the cloud has also been considered in the industry. Thomas et al. [71], [72] proposed the Server and Network-assisted DASH (SAND) architecture to overcome the fact that the client-driven approach of DASH left less control to the network and service providers. RESA [73] employs a learning-based ABR proxy to make a suitable decision for each client. To this end, we believe that deploying an ABR service on the server or edge is also a practical way for today's device and network environments.

VIII. Related Work

A. ABR schemes

Client-based ABR algorithms [2] are mainly organized into two types: model-based and learning-based. The model-based algorithm uses heuristics to construct a model, as the learning-based method adopts deep learning to generalize a strategy from tabular rasa.

Model-based. The development of ABR algorithms begins with the idea of predicting throughput. PANDA [6] predicts the future throughput for eliminating the ON-OFF steady issue. FESTIVE [7] estimates future throughput via the harmonic mean of the throughput measured for the past chunk downloads. However, due to the lack of throughput estimation method currently, these approaches still result in poor ABR performance. Meanwhile, most video client leverages a playback buffer to store the video content downloaded from the server temporarily. BBA [8] proposes a linear criterion threshold to control the available playback buffer size. BOLA [9] turns the ABR problem into a utility maximization problem and solve it by using the Lyapunov function. However, the buffer-based approach fails to tackle the long-term bandwidth fluctuation problem. Hence, mixed model-based approaches, e.g., MPC [10], select bitrate for the next chunk by adjusting its throughput discount factor based on past prediction errors and estimating its playback buffer size. Nevertheless, these approaches require careful tuning because they rely on parameters that are quite sensitive to network conditions, resulting in poor performance in unexpected network environments. What's more, Akhtar et al. [17] propose an auto-tuning method to improve model-based ABR's performance.

Learning-based: Several attempts have been made to optimize the ABR algorithm based on the RL method due to the difficulty of tuning mixed approaches for handling different network conditions. Pensieve [12] is a system that leverages RL to select bitrate for future video chunks. D-DASH [13] uses the Deep Q-learning method to perform a comprehensive evaluation. Tiyuntsong optimizes itself towards a rule or a specific reward via the competition with two agents under the same network condition [15].

In general, existing ABR algorithms seldom consider the time vary of network status. In particular, learning-based ABR schemes fail to tackle the sample efficiency problem.

B. Imitation Learning meets Networking

Imitation learning [74], [75] is the process by which an agent tries to learn how to perform a certain task using information generated by another, often more expert agent performing that same task. Till now, imitation learning has been widely used in various fields including networking scheduling and network congestion control schemes. Tang et al. [76] propose a real-time deep learning-based intelligent network traffic control method to represent the considered Wireless Mesh Network (WMN) backbone via imitation learning. Indigo [77] uses DAgger [38] to train a congestion-control NN scheme in the offline network emulator.

C. Lifelong learning methods

Lifelong learning (or namely continual learning and incremental learning) has become one of the research hotspots for tackling catastrophic forgetting problem. Recently, several approaches have been proposed to extend the loss function with additional terms for guaranteeing the performance on previous tasks, e.g., Learning without Forgetting (LwF) [23] uses outputs of the old models as soft targets of old tasks. These soft targets are considered as a substitute for the data of previous tasks, which cannot be accessed in lifelong learning settings. Another kind of lifelong learning methods estimate the importance of model parameters with specifically designed mechanisms and apply an individual penalty for each previous task, including Elastic Weights Consolidation (EWC) [78], Synaptic Intelligence (SI) and Memory Aware Synaptic (MAS) [79]. However, such lifelong learning methods usually suffer from a key issue: one method that performs well in some experimental settings may fail in others [37].

IX. Conclusion

In this work, we propose Comyco, a learning-based ABR system which aim to thoroughly improve the performance of learning-based algorithm. In general, Comyco makes the contributions as follows: First, we construct Comyco as a video quality-based ABR system, including its NN architectures, datasets and QoE metrics. With trace-driven emulation and real-world deployment. Second, 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. Third, through data-driven analysis we find Comyco should be updated continually over time. Thus, we present lifelong learning-based Comyco, aiming to improve its adaption on network status. Massive of experimental results show that Comyco significantly improves the performance, effectively accelerates the training process, and achieves lifelong training on the entire session.

Additional research will focus on i) applying exogenous features (i.e., date, hour, etc) into the NN, ii) deploying feasible personalized Comyco framework, iii) demystifying the key principle of Comyco, as well as iv) developing a practical scheme for low latency live streaming scenario.

Acknowledgement

We thank the anonymous reviewer for the valuable feedback. This work was supported by the National Key R&D Program of China (No. 2018YFB1003703), NSFC under Grant 61521002, Beijing Key Lab of Networked Multimedia, and Kuaishou-Tsinghua Joint Project (No. 20192000456). This paper extends [80] by adding lifelong learning methods which significantly improves learning-based ABR algorithms to work in practice.

References

[1] Cisco, "Cisco visual networking index: Forecast and methodology, 2016-2021," 2017. [Online]. Available: https://www.cisco.com/c/dam/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.pdf

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

[3] M. Licciardello, M. Grüner, and A. Singla, "Understanding video streaming algorithms in the wild," *arXiv preprint arXiv:2001.02951*, 2020.

[4] T.-Y. Huang, C. Ekanadham, A. J. Berglund, and Z. Li, "Hindsight: Evaluate video bitrate adaptation at scale," in *Proceedings of the 10th ACM Multimedia Systems Conference*, ser. MMSys '19. New York, NY, USA: ACM, 2019, pp. 86–97. [Online]. Available: http://doi.acm.org/10.1145/3304109.3306219

[5] Y. Sun and et al., "Cs2p: Improving video bitrate selection and adaptation with data-driven throughput prediction," in *SIGCOMM 2016*. ACM, 2016, pp. 272–285.

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

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

[8] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson, "A buffer-based approach to rate adaptation: Evidence from a large video streaming service," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 187–198, 2015.

[9] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman, "Bola: Near-optimal bitrate adaptation for online videos," in *INFOCOM 2016, IEEE*. IEEE, 2016, pp. 1–9.

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

[11] K. Spiteri, R. Sitaraman, and D. Sparacio, "From theory to practice: improving bitrate adaptation in the dash reference player," in *Proceedings of the 9th MMSys*. ACM, 2018, pp. 123–137.

[12] H. Mao, R. Netravali, and M. Alizadeh, "Neural adaptive video streaming with pensieve," in *Proceedings of the 2017 ACM SIGCOMM Conference*. ACM, 2017, pp. 197–210.

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

[14] S. Sengupta, N. Ganguly, S. Chakraborty, and P. De, "Hotdash: Hotspot aware adaptive video streaming using deep reinforcement learning," in *2018 IEEE 26th International Conference on Network Protocols (ICNP)*. IEEE, 2018, pp. 165–175.

[15] T. Huang, X. Yao, C. Wu, R.-X. Zhang, and L. Sun, "Tiyuntsong: A self-play reinforcement learning approach for abr video streaming," *arXiv preprint arXiv:1811.06166*, 2018.

[16] "Kuaishou," 2019. [Online]. Available: https://www.kuaishou.com

[17] Z. Akhtar and et al., "Oboe: auto-tuning video abr algorithms to network conditions," in *SIGCOMM 2018*. ACM, 2018, pp. 44–58.

[18] T. Huang, R.-X. Zhang, C. Zhou, and L. Sun, "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, 2018, pp. 1208–1216.

[19] R. Mendonca, A. Gupta, R. Kralev, P. Abbeel, S. Levine, and C. Finn, "Guided meta-policy search," *arXiv preprint arXiv:1904.00956*, 2019.

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

[21] Z. Duanmu, A. Rehman, and Z. Wang, "A quality-of-experience database for adaptive video streaming," *IEEE Transactions on Broadcasting*, vol. 64, no. 2, pp. 474–487, June 2018.

[22] T. Osa, J. Pajarinen, G. Neumann, J. A. Bagnell, P. Abbeel, J. Peters *et al.*, "An algorithmic perspective on imitation learning," *Foundations and Trends® in Robotics*, vol. 7, no. 1-2, pp. 1–179, 2018.

[23] Z. Li and D. Hoiem, "Learning without forgetting," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[24] "Http live streaming," https://developer.apple.com/streaming/, 2019.

[25] "Dash industry forum — catalyzing the adoption of mpeg-dash," 2019. [Online]. Available: https://dashif.org/

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

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

[28] A. Bentaleb, A. C. Begen, and R. Zimmermann, "Sdndash: Improving qoe of http adaptive streaming using software defined networking," in *Proceedings of ACM MultiMedia 2016*. ACM, 2016, pp. 1296–1305.

[29] F. Y. Yan, H. Ayers, C. Zhu, S. Fouladi, J. Hong, K. Zhang, P. Levis, and K. Winstein, "Learning in situ: a randomized experiment in video streaming," in *17th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 20)*, 2019.

[30] Mao, "hongzimao/pensieve," Jul 2017. [Online]. Available: https://github.com/hongzimao/pensieve

[31] Z. Wang, "Video qoe: Presentation quality vs. playback smoothness," Jul 2017. [Online]. Available: https://www.ssimwave.com/science-of-seeing/video-quality-of-experience-presentation-quality-vs-playback-smoothness/

[32] Y. Qin, S. Hao, K. R. Pattipati, F. Qian, S. Sen, B. Wang, and C. Yue, "Abr streaming of vbr-encoded videos: characterization, challenges, and solutions," in *Proceedings of CoNeXT 2018*. ACM, 2018, pp. 366–378.

[33] Z. Duanmu, K. Ma, and Z. Wang, "Quality-of-experience of adaptive video streaming: Exploring the space of adaptations," in *Proceedings of the 25th ACM international conference on Multimedia*. ACM, 2017, pp. 1752–1760.

[34] H. Mao, S. B. Venkatakrishnan, M. Schwarzkopf, and M. Alizadeh, "Variance reduction for reinforcement learning in input-driven environments," *international conference on learning representations*, 2019.

[35] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[36] M. Laskey, J. Lee, R. Fox, A. Dragan, and K. Goldberg, "Dart: Noise injection for robust imitation learning," *arXiv preprint arXiv:1703.09327*, 2017.

[37] R. Kemker, M. McClure, A. Abitino, T. L. Hayes, and C. Kanan, "Measuring catastrophic forgetting in neural networks," in *Thirty-second AAAI conference on artificial intelligence*, 2018.

[38] S. Ross, G. Gordon, and D. Bagnell, "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*, 2011, pp. 627–635.

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

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

[41] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[42] X. Yao, T. Huang, C. Wu, R.-X. Zhang, and L. Sun, "Adversarial feature alignment: Avoid catastrophic forgetting in incremental task lifelong learning," *Neural computation*, vol. 31, no. 11, pp. 2266–2291, 2019.

[43] J. Benesty, J. Chen, Y. Huang, and I. Cohen, "Pearson correlation coefficient," in *Noise reduction in speech processing*. Springer, 2009, pp. 1–4.

[44] P. G. Pereira, A. Schmidt, and T. Herfet, "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, 2018, pp. 43–48.

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

[46] T. Abar, A. B. Letaifa, and S. El Asmi, "Machine learning based qoe prediction in sdn networks," in *2017 13th International Wireless Communications and Mobile Computing Conference (IWCMC)*. IEEE, 2017, pp. 1395–1400.

[47] A. Aaron, Z. Li, M. Manohara, J. Y. Lin, E. C.-H. Wu, and C.-C. J. Kuo, "Challenges in cloud based ingest and encoding for high quality streaming media," in *2015 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2015, pp. 1732–1736.

[48] A. Rehman, K. Zeng, and Z. Wang, "Display device-adapted video quality-of-experience assessment," in *Human Vision and Electronic Imaging XX*, vol. 9394. International Society for Optics and Photonics, 2015, p. 939406.

[49] Z. Duanmu, W. Liu, D. Chen, Z. Li, Z. Wang, Y. Wang, and W. Gao, "A knowledge-driven quality-of-experience model for adaptive streaming videos," *arXiv preprint arXiv:1911.07944*, 2019.

[50] A. Rehman and Z. Wang, "Perceptual experience of time-varying video quality," in *2013 Fifth International Workshop on Quality of Multimedia Experience (QoMEX)*. IEEE, 2013, pp. 218–223.

[51] "Youtube," 2019. [Online]. Available: https://www.youtube.com

[52] FFmpeg, "Ffmpeg." [Online]. Available: http://ffmpeg.org/

[53] GPAC, "Mp4box." [Online]. Available: https://gpac.wp.imt.fr/mp4box/

[54] T. Huang, "Comyco video description dataset," https://github.com/godka/Comyco-Video-Description-Dataset/, 2020.

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

[56] Y. Tang, "Tf. learn: Tensorflow's high-level module for distributed machine learning," *arXiv preprint arXiv:1612.04251*, 2016.

[57] D. M. Beazley *et al.*, "Swig: An easy to use tool for integrating scripting languages with c and c++." in *Tcl/Tk Workshop*, 1996, p. 43.

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

[59] T. Huang, "Comyco," https://github.com/thu-media/comyco/, 2020.

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

[61] Usc-Nsl, "Usc-nsl/oboe," Oct 2018. [Online]. Available: https://github.com/USC-NSL/Oboe

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

[63] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," *arXiv preprint arXiv:1611.06440*, 2016.

[64] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "Shufflenet v2: Practical guidelines for efficient cnn architecture design," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 116–131.

[65] H. Mao, S. Chen, D. Dimmery, S. Singh, D. Blaisdell, Y. Tian, M. Alizadeh, and E. Bakshy, "Real-world video adaptation with reinforcement learning," in *ICML 2019 Workshop*, 2019.

[66] S. Akhshabi, A. C. Begen, and C. Dovrolis, "An experimental evaluation of rate-adaptation algorithms in adaptive streaming over http," in *Proceedings of the second annual ACM conference on Multimedia systems*. ACM, 2011, pp. 157–168.

[67] I. Sodagar, "The mpeg-dash standard for multimedia streaming over the internet," *IEEE multimedia*, vol. 18, no. 4, pp. 62–67, 2011.

[68] D. Smilkov, N. Thorat, Y. Assogba, A. Yuan, N. Kreeger, P. Yu, K. Zhang, S. Cai, E. Nielsen, D. Soergel *et al.*, "Tensorflow. js: Machine learning for the web and beyond," *arXiv preprint arXiv:1901.05350*, 2019.

[69] Z. Meng, J. Chen, Y. Guo, and M. Xu, "Pitree: Practical implementation of abr algorithms using decision trees," in *2019 ACM Multimedia Conference on Multimedia Conference*. ACM, 2019.

[70] A. Dethise, M. Canini, and S. Kandula, "Cracking open the black box: What observations can tell us about reinforcement learning agents," in *Proceedings of the 2019 Workshop on Network Meets AI & ML*. ACM, 2019, pp. 29–36.

[71] E. Thomas, M. van Deventer, T. Stockhammer, A. C. Begen, and J. Famaey, "Enhancing mpeg dash performance via server and network assistance," 2015.

[72] E. Thomas, M. van Deventer, T. Stockhammer, A. C. Begen, M.-L. Champel, and O. Oyman, "Applications and deployments of server and network assisted dash (sand)," 2016.

[73] Y. Wang, H. Wang, J. Shang, and H. Tuo, "Resa: A real-time evaluation system for abr," in *2019 IEEE International Conference on Multimedia and Expo (ICME)*. IEEE, 2019, pp. 1846–1851.

[74] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne, "Imitation learning: A survey of learning methods," *ACM Computing Surveys (CSUR)*, vol. 50, no. 2, p. 21, 2017.
[75] F. Torabi, G. Warnell, and P. Stone, "Recent advances in imitation learning from observation," *arXiv preprint arXiv:1905.13566*, 2019.
[76] F. Tang, B. Mao, Z. M. Fadlullah, N. Kato, O. Akashi, T. Inoue, and K. Mizutani, "On removing routing protocol from future wireless networks: A real-time deep learning approach for intelligent traffic control," *IEEE Wireless Communications*, vol. 25, no. 1, pp. 154–160, February 2018.
[77] F. Y. Yan, J. Ma, G. D. Hill, D. Raghavan, R. S. Wahby, P. Levis, and K. Winstein, "Pantheon: the training ground for internet congestion-control research," in *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)*, 2018, pp. 731–743.
[78] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska *et al.*, "Overcoming catastrophic forgetting in neural networks," *Proceedings of the national academy of sciences*, p. 201611835, 2017.
[79] R. Aljundi, F. Babiloni, M. Elhoseiny, M. Rohrbach, and T. Tuytelaars, "Memory aware synapses: Learning what (not) to forget," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 139–154.
[80] T. Huang, C. Zhou, R.-X. Zhang, C. Wu, X. Yao, and L. Sun, "Comyco: Quality-aware adaptive video streaming via imitation learning," *arXiv preprint arXiv:1908.02270*, 2019.

PLACE PHOTO HERE

Tianchi Huang received his M.E degree in the Department of Computer Science and Technology in Guizhou University in 2018. Currently he is a Ph.D student in the Department of Computer Science and Technology at Tsinghua University, advised by Prof. Lifeng Sun. His research work focuses on the multimedia network streaming, including transmitting streams, and edge-assisted content delivery. He has been the reviewer for IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY and IEEE TRANSACTIONS ON MULTIMEDIA.

PLACE PHOTO HERE

Chao Zhou received his Ph.D. degree from the Institute of Computer Science and Technology, Peking University, Beijing, China, in 2014. He has been with Beijing Kuaishou Technology Co., Ltd. as an Algorithm Scientist. Before joining Kuaishou, he was a Senior Research Engineer with the Media Technology Lab, CRI, Huawei Technologies CO., LTD, Beijing, China. Dr. Zhou's research interests include HTTP video streaming, joint source-channel coding, and multimedia communications and processing. He has been the reviewer for IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY , IEEE TRANSACTIONS ON MULTIMEDIA , IEEE TRANSACTIONS ON WIRELESS COMMUNICATION and so on. He received Best Paper Award presented by IEEE VCIP 2015, and Best Student Paper Awards presented by IEEE VCIP 2012.

PLACE PHOTO HERE

Xin Yao is currently a Ph.D. candidate in the Department of Computer Science and Technology at Tsinghua University, advised by Prof. Lifeng Sun. He received his bachelor's degree from the Department of Computer Science and Technology at Tsinghua University in 2016. His research interests focus on federated learning, lifelong/continual learning, and transfer learning.

PLACE PHOTO HERE

Rui-Xiao Zhang received his B.E degree in Electronic Engineering Department in Tsinghua University in 2017. Currently, he is pursuing his Ph.D candidate in Department of Computer Science and Technology, Tsinghua University, China. His research interests lie in the area of content delivery networks, the optimization of multimedia streaming and reinforcement learning. He received Best Student Paper Awards presented by ACM Multimedia System 2019 Workshop.

PLACE PHOTO HERE

Chenglei Wu received the Master degrees in Tsinghua University. He is currently a Ph.D with the Computer Science and Technology Department of Tsinghua University. His research interests focus on 360 video streaming, adaptive video streaming and routing.

PLACE PHOTO HERE

Bing Yu , the leader of the audio and video technology of Kuaishou, graduated from Tsinghua University, has many years of experience in the video and streaming media industry. He is good at using advanced Internet technology and data-driven concepts to optimize the system for providing users with the best QoE. Prior to joining Kuaishou, he led video technology and infrastructure teams at multinational companies such as Hulu and FreeWheel.

PLACE PHOTO HERE

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

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

APPENDIX A
SUMMARY OF STATISTICS FROM THE DATASET

Type	Dataset Name
Network Traces	FCC [27], HSDPA [26], Oboe [61]
Lifelong Training Traces	Kwai (§II)
Video Description Datasets	CVDD [54]
QoE Database for ABR	SQoEIII [21]