Received March 30, 2022, accepted April 26, 2022, date of publication May 16, 2022, date of current version May 23, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3175209

ABRaider: Multiphase Reinforcement Learning for Environment-Adaptive Video Streaming

WANGYU CHOI[1], JIASI CHEN[2], (Member, IEEE), AND JONGWON YOON[1], (Member, IEEE)
[1]Department of Computer Science and Engineering, Hanyang University, Ansan 15588, South Korea
[2]Department of Computer Science and Engineering, University of California at Riverside, Riverside, CA 92521, USA

Corresponding author: Jongwon Yoon (jongwon@hanyang.ac.kr)

This work was supported in part by the Basic Science Research Program through the National Research Foundation of South Korea (NRF) by the Ministry of Education under Grant NRF-2022R1A2C1008743; and in part by the Ministry of Science and ICT (MSIT), South Korea, under the Grand Information and Communication Technology Research Center Support Program supervised by the Institute for Information and Communications Technology Planning and Evaluation (IITP), under Grant IITP-2020-101741.

ABSTRACT HTTP-based video streaming technology is widely used in today's video delivery services. The streaming solution uses the adaptive bitrate (ABR) algorithm for better video quality and user experience. Despite many efforts to improve the quality of experience (QoE), it is very challenging for ABR algorithms to guarantee high QoE to all users in various environments. The video streaming circumstances in the real world have become even more complicated by the proliferation of mobile devices, high-quality content, and heterogeneous configurations of video players. Many ABR algorithms aim to find monotonous strategies that generally perform well without focusing on the complexity of the environments, which can degrade performance. In this paper, we propose *ABRaider* that guarantees high QoE to all users in a variety of environments in the real world while being *generalized* with multiple strategies and *specialized* in each user's environment. In *ABRaider*, we propose multi-phase RL consisting of offline and online phases. In the offline phase, *ABRaider* integrates the strengths of the ABR algorithms and develops policies suitable for various environments. In the online phase, *ABRaider* focuses on specializing in the environments of individual users by leveraging the computational power of the clients. Experiment results show that *ABRaider* outperforms existing solutions in various environments, achieving 19.9% and 42.2% QoE improvement in VoD and live streaming, respectively.

INDEX TERMS Adaptive bitrate algorithm, federated learning, quality of experience, reinforcement learning, video streaming.

I. INTRODUCTION

With the increase of network bandwidth and the spread of high-quality videos, users tend to prefer video-based information over text-based one. The video data becomes the majority (65%) of worldwide mobile downstream and is expected to continue to increase [1], [2]. Unfortunately, the current network infrastructures are unable to handle this massive growth seamlessly and meet the user's demands. Users want high-quality video streaming, and service providers are constantly striving to improve the quality. Nevertheless, providing users with high QoE (quality of experience) is an underlying challenge because network bandwidth is limited, shared, and unpredictable.

Many video streaming solutions utilize ABR (adaptive bitrate) approaches to provide satisfactory services [3]. The ABR algorithm adjusts the video bitrate in order to optimize video quality on the client-side. Despite these efforts, the proliferation of mobile devices, the demand for high-quality content, and the heterogeneous configurations of video players make the environment even complicated. Specifically, mobile devices induce various mobility trajectories, resulting in unpredictable bandwidth estimation. Most videos have different/wide range of bitrate sets and segment lengths (e.g., 1–15 seconds [4]) depending on the encoder. ABR algorithm has to carefully consider the configurations of video player to provide high QoE. For example, Youtube mobile app or Youtube Live requires a conservative and sensitive strategy due to the small buffer, otherwise it is vulnerable to stall events and low-quality. Given all these circumstances, it is very difficult for the ABR algorithm to provide high QoE to all users in the real world.

Recently, ABR algorithms using various approaches, such as rule-based and machine learning (ML)-based, have been

The associate editor coordinating the review of this manuscript and approving it for publication was Victor S. Sheng.

proposed to deliver users with high QoE. The rule-based ABR algorithms use heuristics that reduce rebuffering, improve the utility, and select bitrate by maximizing QoE metric [5]–[8]. However, we have confirmed that each rule-based algorithm has a preferred environment with a monotonous strategy (details in Section III-A). They are fine-tuned in specific configurations and do not guarantee high QoE in various environments. ML-based approaches are classified into reinforcement learning (RL)-based and supervised learning (SL)-based. The RL-based ABR algorithms use neural networks to train bitrate adaptation policy [9] and they achieve significant performance gains in terms of QoE, but their improvements are limited in certain environments. Their performance is tightly coupled with the training environment [10]. SL-based ABR algorithms [11], [12] predict bandwidth more accurately by learning network traces instead of harmonic mean or moving average of past segment download times. However, since they still work with MPC [7], which is a rule-based algorithm, they naturally do not adapt to a variety of environments.

We set two keys to unlock the ABR algorithm to provide high QoE to the users; *generalization* and *specialization*. The ABR algorithm should be generalized with multiple strategies for enhanced performance and specialized in consideration of the individual user's environment. In this paper, we introduce *ABRadier*, a generalized and individually specialized ABR algorithm for the users in various environments. *ABRaider* addresses two key challenges: (i) how do we generate an ABR algorithm that has multiple strategies for the different configurations? (ii) how does the ABR algorithm guarantee high QoE for all users in the real world? To answer the above questions, we propose *multi-phase RL* that consists of the *offline* and the *online phases* to overcome the shortcomings of traditional RL. The multi-phase RL brings several advantages. It serves as a key component in *generalizing* the ABR algorithm to users under various circumstances and *specializing* in each user's specific environment. In particular, the offline phase allows *ABRaider* to explore the strengths of existing ABR algorithms and develop different strategies for each environment. At the end of the offline phase, *ABRaider* achieves generalized performance in various environments by retaining multiple strategies of the algorithms. Leveraging the ever-increasing clients' computational power, we enable *ABRaider* to continuously train under clients' unique/specific environments in a distributed manner during the online phase. *ABRaider* aims to be *specialized* in each user's environment by fine-tuning the generalized model of the offline phase. To this end, we have developed a publicly accessible online training website. Whenever users access it, *ABRaider* is distributed individually and evolves into a model tailored to the user-specific environment through a video streaming experience.

We implement *ABRaider* in dash.js [13], a standard MPEG-DASH player, and use publicly available network traces to create a variety of training/test environments. These datasets are collected from Wi-Fi, LTE, and 5G networks. *ABRaider's* performance is compared with the state-of-the-art ABR algorithms in various configurations. Experiment results show that *ABRaider* improves the average QoE by 19.9% and 42.2% compared to the best ABR algorithm in VoD and live streaming, respectively.

The rest of this paper is organized as follows. In Section II, we discuss the related work. We introduce several challenges of the ABR algorithms and motivate the necessity of a new approach in Section III. Prior to designing our system, Section IV describes key design choices. Then, we propose *ABRaider* and explain both the training scheme and the multi-phase RL in Section V. Section VI evaluates the performance of *ABRaider* compared with other ABR algorithms. Finally, we conclude the paper in Section VIII.

II. RELATED WORK

The recently proposed ABR algorithms can be categorized into two classes according to the approach.

A. RULE-BASED

The rule-based ABR algorithm aims to high quality and low rebuffering in order to provide satisfaction to users. Festive [5] uses the harmonic mean of the throughput over the past 20 segments to predict the future bandwidth and then determines the maximum bitrate it can sustain. BBA [6] only uses the playback buffer occupancy to determine the bitrate. The larger the buffer size, the higher the bitrate. BBA reduces the rebuffering rate by 10-20% compared to Netflix's ABR algorithm. To overcome the shortcomings of the above heuristic-based ABR algorithms, several approaches define QoE/utility metrics and design ABR algorithms that theoretically maximize the metrics. Specifically, MPC [7] defines a QoE metric (quantifying the QoE value delivered to users), and uses a model predictive control algorithm to select a bitrate at which QoE can be maximized. BOLA [8], [14] uses Lyapunov optimization to maximize two metrics; bitrate utility (i.e., quality) and a fraction of time spent not rebuffering. The above-mentioned approaches made significant performance improvements by maximizing the metrics directly.

B. ML-BASED

ABR algorithms using a machine learning approach have been proposed to address the challenge of bandwidth prediction in variable networks [9], [11], [12], [15]. ML-based algorithms are classified into reinforcement learning-based [9], [15] and supervised learning-based approaches [11], [12]. Pensieve [9] used reinforcement learning to directly train the policy of the neural network to design ABR algorithm using a video streaming simulator, which improved QoE by 15-25% compared to MPC. Zhang *et al.* [15] have proposed OnRL, online reinforcement learning framework. Like *ABRaider*, they employ federated learning in which the clients train the policy in a distributed manner. However, the target system of OnRL is limited to the live video telephony service such as FaceTime, Zoom and Skype.

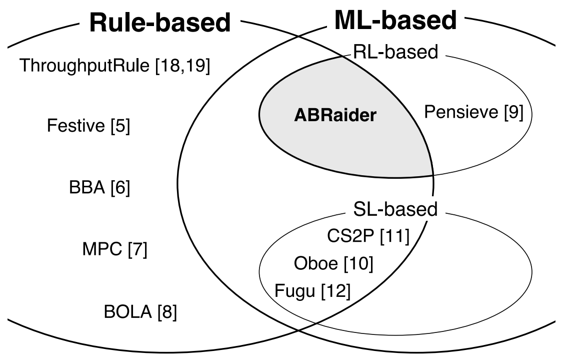


FIGURE 1. ***ABRaider*** **employs an RL-based approach while retaining the advantages of traditional rule-bases algorithms.**

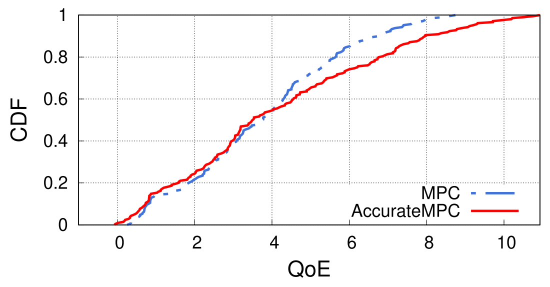


FIGURE 2. **Performance comparison between MPC and AccurateMPC in various environments. The average QoE of MPC and AccurateMPC are 3.8 and 4.1, respectively.**

There are several studies that explicitly predict future bandwidth using supervised learning instead of directly training the policy (i.e., RL). The authors in [11] analyzed throughput characteristics using datasets with more than 20 Million sessions. They developed CS2P, a throughput prediction system using Hidden-Markov-Model. The combination of CS2P and MPC achieved 3.2% QoE improvement over MPC in practice. Similarly, the authors in [12] used supervised learning to train a transmission time predictor (TTP) for the future segments on a public video streaming website. As a result, Fugu, an ABR algorithm that is a combination of TTP and MPC, outperforms existing schemes in their system in terms of average SSIM and stall rates.

C. POSITION OF OUR WORK

As mentioned above, ABR algorithms to improve user experience have been proposed along with various approaches. Figure 1 illustrates the existing algorithms' approaches and the position of our work. Recently, ML-based solutions are emerged to cope with dynamic networks because rule-based algorithms are difficult to have different strategies due to the limited number of rules. The SL-based approach improves the performance of one of the rule-based algorithm's modules (e.g., bandwidth predictor), which struggles to handle a variety of real-world environments (details in Section III-A). RL-based schemes learn the policy directly to improve QoE metrics in a trial-and-error manner, but are vulnerable if training methods or environments (e.g., simulator and network traces) are not carefully managed [10], [12]. In this paper, we adopt RL with the potential for various strategies in the real world environment and leverage existing rule-based algorithms to overcome the limitations of traditional RL. Our RL training scheme learns policies from scratch, something new and undiscovered, while retaining all the strengths of existing rule-based algorithms.

III. CHALLENGE AND MOTIVATION

We discuss the motivations for designing a better ABR algorithm, then describe goals of *ABRaider*.

A. CHALLENGES OF THE ABR ALGORITHMS

Several challenges arise when operating the ABR algorithm in various real-world environments. We explain in detail the challenges associated with the strategy and generalization of ABR algorithm.

1) LIMITATIONS OF A MONOTONOUS STRATEGY

Rule-based algorithms such as rate-based and buffer-based are usually limited to have a single strategy. This is because they use only one-dimensional information to determine the bitrate, ignoring other observations/circumstances. Hybrid approaches such as MPC, which are more sophisticated algorithms, have relatively diverse strategies. For example, they can make aggressive decisions when the bandwidth is stable and sufficient even if the buffer occupancy is low. However, these are still insufficient to handle the diverse environments in the real world, despite perfect bandwidth predictions.

To investigate this, we conducted a performance comparison using MPC, which the SL-based schemes (e.g., Fugu [12]) relies. We configured 135 environments with respect to the network bandwidth range, degree of variation, video bitrate range, and the segment length (details in Section VI). In each environment, QoE is calculated using Equations (3) and (4) in Section V-A. For comparison with MPC, we introduce AccurateMPC, a variant of MPC, which uses the actual throughputs of the network traces instead of the bandwidth prediction module. Intuitively, it is the *upper bound of SL-based algorithm*. Figure 2 shows the CDF of QoE for MPC and AccurateMPC. We clearly see that even if the future bandwidth can be predicted perfectly, AccurateMPC is not the best in all circumstances. Specifically, its average QoE is about 0.3 higher than the MPC, and is somewhat lower in the 46% of environments despite perfect bandwidth estimates. In AccurateMPC, unnecessary bitrate switching occurs when the network fluctuates greatly or when video segments are long, whereas in MPC, the bandwidth prediction module has the effect of mitigating the bandwidth fluctuations. In other words, SL-based ABR algorithms aim to provide high QoE on average with a monotonous strategy.

Based on these results, we argue that multiple strategies are necessary for the ABR algorithm to yield robust performance in various environments.

2) GENERALIZATION IN THE REAL WORLD

Although machine learning approaches (e.g., RL-based and SL-based) have been adopted to address the above-mentioned

TABLE 1. The rank frequency for ABR algorithms in 135 environments.

Algorithm / Rank	1st	2nd	3rd	4th	5th	6th	7th
ThroughputRule [18], [19]	0	0	1	8	53	60	13
Festive [5]	1	7	32	39	40	15	1
BBA [6]	42	16	17	13	5	4	38
MPC [7]	50	37	31	12	3	2	0
BOLA [8]	1	1	12	25	21	15	60
Pensieve [9]	19	16	13	17	10	37	23
Oboe+MPC [10]	22	58	29	21	3	2	0

complexity, there are still unresolved problems. They require a large-size model (e.g., deep neural networks) with a lot of data in order to generalize it in the real world. The larger the model, the more strategies it can contain, allowing it to cover a wide variety of environments. This not only imposes a burden on the client or server managing the model, but also requires sophisticated techniques for training and incurs high computational costs [16], [17]. With a limited model size, it only converges to a model that performs well on average.

In summary, it is very difficult to generalize rule-based and machine learning-based ABR algorithms to cover a wide range of different environments in the real world. We envision an ABR algorithm that can provide high-quality services to all users in various environments by training it in user-level environments with only a small-sized neural network.

B. MOTIVATIONS OF ABRaider

1) FRAGMENTATION OF THE STRENGTHS OF ALGORITHMS

As mentioned above, many ABR algorithms have limitations due to a monotonous strategy, however each one has enough strengths to take advantage of. To this end, we examined the performance of various algorithms in the same environments as in the previous experiment. Table 1 shows the rank frequency for each ABR algorithm. We can confirm that ABR algorithm has its own preferred environment where it provides the best performance comparing to others. We refer to this phenomenon as *strength fragmentation of ABR algorithm*. For example, BBA is ranked 1st in 42 environments and has the lowest performance in 40 environments (ranked 7th). BBA prefers a shorter segment (e.g., 2 seconds) and a relatively high bandwidth in the bitrate set. Furthermore, it is robust on the fluctuation of bandwidth. The rate-based algorithms, such as ThroughputRule and Festive, can quickly select an appropriate bitrate regardless of the buffer (e.g., startup phase). MPC or MPC-variant (e.g., Oboe+MPC) combines the strengths of both approaches to provide good QoE on average (e.g., steady phase).

In summary, the existing algorithms have the strength of being able to perform better in certain situations, this motivates us designing a better ABR algorithm utilizing multiple strategies. Toward this, ABRaider leverages the strengths of existing algorithms to maximize generalization in variety of environments.

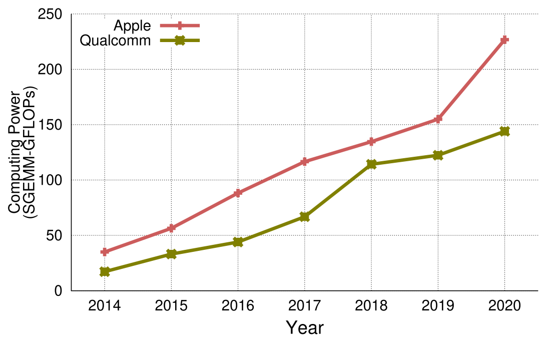


FIGURE 3. Growth of mobile GPUs [20], [21].

2) UNDER-UTILIZATION OF CLIENT's COMPUTING POWER

Mobile devices such as smartphones are the main platform for streaming video services on the client side [22]. They have the powerful computing power of CPUs and GPUs, which is expected to continue to grow. Figure 3 shows the increasing computing power of mobile GPUs over time. Most recent mobile devices have hardware engines (i.e., neural engine) dedicated to neural processing, but their performance is not even exploited. For instance, many ABR algorithms use fixed control rules (CPUs). ML-based algorithms such as Pensieve and Fugu determines bitrates on the server side.

In summary, existing bitrate adaptation schemes under-utilize the powerful computing power of mobile devices. ABRaider *leverages high-performance GPUs to train on the client side, and hence, specializes in individual user environments.*

C. GOAL OF ABRAIDER

The challenges of the ABR algorithm mentioned above motivate us to design and implement an ABR algorithm that can provide satisfactory streaming quality to users in various environments. We set *ABRaider*'s goals as follows: (i) ABR algorithm should be *generalized* in a variety of environments with multiple strategies, (ii) ABR strategies should be *specialized* for each user, not providing generalized performance after deployment. Toward this, we utilize reinforcement learning technique and introduce multi-phase RL to overcome the limitations of traditional RL. It combines the advantages of existing schemes and adapts *ABRaider* with multiple strategies suitable for various environments (i.e., generalization). Furthermore, it allows *ABRaider* to be *specialized* in real-world users through client-level federated learning in practice.

We aim to generalize and specialize the ABR algorithm for users in a variety of environments. In ABRaider*'s design, we employ multi-phase RL to overcome the limitations of traditional RL and achieve the goals mentioned above.*

IV. KEY DESIGN CHOICES

This section describes key design choices for *ABRaider*. In particular, an in-depth exploration of when and how to train is necessary because we consider a data-driven learning framework for the ABR algorithm.

A. SL VS. RL

Existing ML-based approaches are divided into SL-based [11], [12] and RL-based [9], [12]. SL-based algorithms typically predict the download times of future segments, which are then used by rule-based algorithms (e.g., MPC) to determine the bitrate. Although more accurate predictions can improve the performance of rule-based algorithms almost to the upper bound, there are obvious limitations (Section III-A). The RL approach, on the other hand, updates the policy that determines the bitrate directly for better QoE. It has more potential than SL-based algorithms. Therefore, we utilize RL to envision a robust ABR algorithm that adapts to various environments.

B. OFFLINE VS. ONLINE

Most existing solutions use only offline or online training. Schemes using offline training [9], [11] may be tightly coupled with the training environment and suffer in various environments. In contrast, solutions that only consider online training [12], [15] may provide a poor user experience at the beginning of training. We address the shortcomings of both approaches by integrating the entire training process into offline and online phases. The offline phase makes the ABR algorithm robust regardless of the environment, while the online phase aims to further improve the model trained in the offline phase.

C. SERVER-SIDE VS. CLIENT-SIDE

Several ML-based ABR algorithms that determine the bitrate on the server-side have some limitations [9], [12]. In an ABR streaming system running on server-side, the client must query the ABR server to determine the bitrate of the next video segment, which incurs additional round trip overhead. This overhead is small, typically 1-100 msec, but cannot be ignored when the network is poor and the client has to constantly download small segments to fill the buffer. Such server-side design cannot guarantee scalability on limited hardware resources. Considering all, we employ a client-side architecture with minimal changes to underlying HTTP-based video streaming. Moreover, we leverage the client's GPUs to ensure scalability without additional overhead.

D. POLICY INITIALIZATION

A starting point is very important in reinforcement learning. The more complex tasks such as real-world problems, the lower the performance of RL using only explicit reward. To address this issue, several studies suggest imitation learning using expert policy [23]–[25]. In bitrate adaptation task, complex environments like the real world are burdensome to train using RL. Moreover, it can provide users a bad experience at the beginning of training (i.e., anomaly in RL policy). The authors in [15] proposed a hybrid learning framework that also uses a rule-based algorithm to avoid the anomaly. Taking into account the complexity of the task, we combine the strengths of existing algorithms (i.e., experts), and adopt them as a baseline. Then, we improve the model through multi-phase RL.

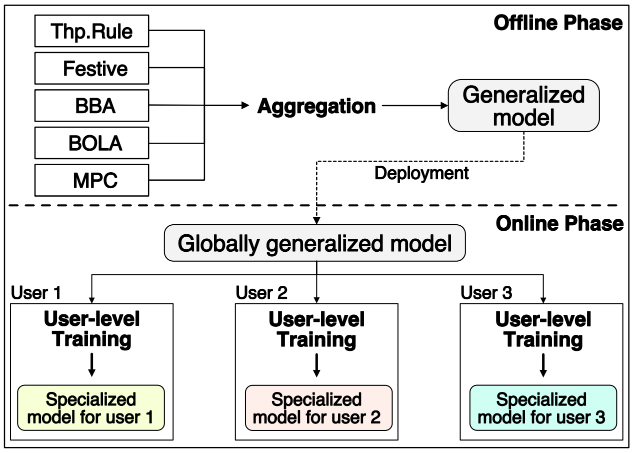


FIGURE 4. The workflow of *ABRaider*.

V. DESIGN

A. OVERVIEW

In this section, we introduce an overview of multi-phase RL in *ABRaider* followed by a detailed exposition of both the offline and online phases. The goal of the multi-phase RL is to develop a robust algorithm that can be applied to a variety of environments. Figure 4 demonstrates the workflow of *ABRaider*. In the offline phase, *ABRaider* takes the strengths of other ABR algorithms and trains the algorithm with various traces in order to incorporate multiple strategies into a single policy. Unlike traditional RL approach that uses only offline training, *ABRaider* includes the online phase. In the online phase, *ABRaider* continues the training process with the trajectories that cannot be handled in the offline phase, and hence is specialized in each environment. In this sense, *ABRaider* is continuously evolving tailored to individual users. We deploy *ABRaider*'s neural network model on a CDN server (publicly available) to facilitate the online phase using crowdsourcing. Specifically, the information (e.g., selected bitrate, buffer occupancy and etc.) generated by streaming video using our model is reflected in the model's updates of clients (i.e., local model). The local models updated in a distributed manner are periodically aggregated into the global model. Over time, the global model generalizes to all users, while the local model specializes in user specifics.

B. OFFLINE PHASE

Existing rule-based and conventional RL-based ABR algorithms provide only local solutions (cannot handle a variety of environments). We found that the ABR algorithm's strengths can be maximized in certain circumstances in Section III-B.

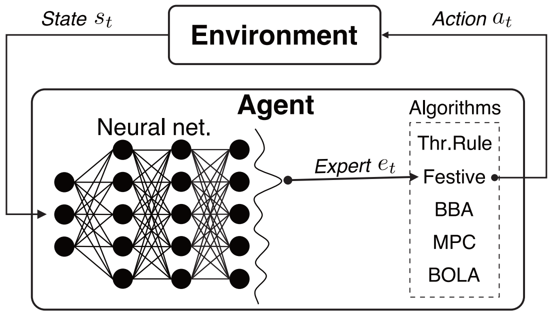


FIGURE 5. The core engine of step 1.

Based on this, we now describe the core engine of the offline phase in multi-phase RL, which alleviates the limitations mentioned above. The offline phase consists of three steps: (i) combining the strengths of existing algorithms, (ii) converting the output space, and (iii) enhancing the algorithm through deep exploration. The offline phase leverages the existing ABR algorithms to generate a robust algorithm that is suitable for various environments.

1) STEP 1: COMBINING THE STRENGTH

In step 1, we aim to design a general solution that can deliver high performance regardless of the environment. Toward this, we first gather the advantages of traditional ABR algorithms, given the fact that each rule-based algorithm performs best in one or more environments. We adopt reinforcement learning to our neural network. The output can be represented as the probability of the existing ABR algorithm $\pi_{\theta^1}(e_t|s_t) = \mathbb{P}[e_t|s_t]$, where θ^1 is the parameters of the neural network, s_t is the observations of the environment (described in Section V-A) and e_t is the expert (i.e., ABR algorithm). The goal of step 1 is to learn the policy $\pi_{\theta^1}(e_t|s_t)$ of selecting the optimal ABR algorithm given the state s_t.

During offline training, we use a simulator instead of configuring a real video streaming player to speed up the training process. We implement a simulator that is similar to the video player used in practice (dash.js). When the simulator receives the bitrates from the agent, it first calculates the download time by obtaining the bandwidth from the network traces. The simulator then consumes the playback buffer occupancy during the segment's download time to mimic the video playback. It also adds the segment length to the playback buffer. While repeating this process, the simulator carefully manages the depletion and overflow of the playback buffer. Figure 5 illustrates an overview of step 1.

The training process in step 1 is as follows. At time step t, the RL agent with the neural network interacts with the environment (simulator) and obtains the trajectory $\{s_0, e_0, a_0, r_1, \cdots, s_{T-1}, e_{T-1}, a_{T-1}, r_T, s_T\}$, where T is the terminal time step, e_t is one of the ABR algorithms selected by the neural network at time t, and a_t is the bitrate determined by e_t. Then we update the neural network using the state-of-the-art policy gradient method PPO [26]. The

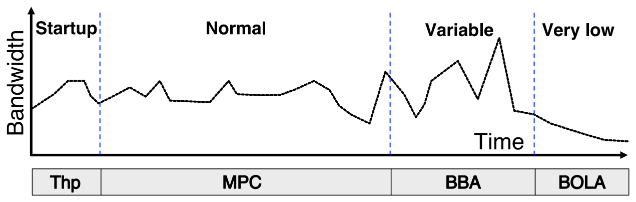


FIGURE 6. Example of offline phase results.

objective function to update the neural network can be written

$$L^{CLIP}(\theta^1) = \hat{\mathbb{E}}\left[\min(r_t(\theta^1)\hat{A}_t, \text{clip}(r_t(\theta^1), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right] \quad (1)$$

where $r_t(\theta)$ is the ratio between the probability of action e_t used in the experience (old) and the new policy $(\frac{\pi_\theta(e_t|s_t)}{\pi_{\theta_{old}}(e_t|s_t)})$. $\hat{A}_t$ is the advantage calculated by [27], and ϵ is one of the hyper-parameters, usually set to 0.1 or 0.2, that has the effect of discarding samples that cause excessively large updates. Equations (3) and (4) are used for the reward signal. Note that different QoE metrics can be used as a reward in *ABRaider's* neural network. By repeating this, the policy $\pi_{\theta^1}(a_t|s_t)$ learns to choose the most suitable ABR algorithm for a given state s_t. However, step 1 has a drawback that the outcome of $\pi_{\theta^1}(a_t|s_t)$ is tightly coupled with the ABR algorithm's performance. To mitigate this, we consider the augmentation of the ABR algorithms by varying the parameters. Table 2 shows the ABR algorithms considered and the variants we used for training.

At the end of step 1, *ABRaider* determines the bitrate while adapts algorithms at the segment level (not the video level) according to changes in the environment. As a result, *ABRaider* outperforms rule-based algorithms in most environments. Figure 6 shows that the algorithm is adapted in terms of bandwidth among various elements constituting the environment. In the startup stage, *ABRaider* determines the bitrate using ThroughputRule, a rate-based algorithm, because the observations are not enough (e.g., buffer occupancy is 0). In the normal stage, it uses MPC and switches to another algorithm when it detects anomalies such as variable or very low bandwidth. When an extreme change in bandwidth is detected, it switches to BBA instead of MPC relying on bandwidth estimation, and when bandwidth is suddenly lowered, it switches to BOLA, which is robust to rebuffering events. Note that this is a typical example of an adaptation based solely on bandwidth observations, where the algorithm and its parameters (Table 2) can be switched by various combinations of observations (e.g., buffer occupancy, segment length, bandwidth and etc).

2) STEP 2: CONVERTING THE OUTPUT SPACE

We design *ABRaider*'s algorithm in step 1, but there is still room for improvement; (i) the policy itself cannot determine the bitrate because the output is one of the ABR algorithms, and (ii) it is impossible to enhance its policy without adjusting the ABR algorithm or adding new ones. To alleviate these

TABLE 2. Parameters configurations for augmentation. Symbol * indicates the default value.

ABR algo.	Set of parameters.
Thp. [18] (α, β)	3,1 | 5,2 | 8,3 | 12,3 | 20,3 | 20,8 | 20,10
Festive [5] (k)	1 | 3 | 5 | 7 | 10 | 15 | 20*
BBA [6] (r, c)	5,20 | 5,25 | 10,20 | 10,25 | 11,27* | 15,27 | 20,27
MPC [7] (d)	1 | 3 | 5* | 7 | 10
BOLA [8] (s, f)	5 | 10* | 15 | 20 | 30

limitations, we convert the output space of $\pi_{\theta^1}(e_t|s_t)$ to a probability of bitrate $\pi_{\theta^2}(a_t|s_t)$. We consider imitation learning (behavior cloning) to clone the policy $\pi_{\theta^1}(e_t|s_t)$ of step 1 into this neural network. Similar to step 1, the agent obtains the following trajectory by interacting with the environment: $\{s_0, e_0, a_0, r_1, \cdots, s_{T-1}, e_{T-1}, a_{T-1}, r_T, s_T\}$. Here we only use the observations from the environment and the bitrate determined by the ABR algorithm (pairs of s_t and a_t). Then we train the neural network to output the bitrate a_t when the state s_t is given (supervised learning). Specifically, given input s_t to the neural network, the KL divergence between the output probability $\pi_{\theta^2}(s_t)$ and the trajectory's action a_t is computed, and gradient descent is applied to the parameters of the neural network with L_2 regularization.

3) STEP 3: ENHANCING THE ALGORITHM

Here we improve the algorithm for environments that cannot be handled by the existing ABR algorithms. We use reinforcement learning to enhance the policy. Once we initialize the neural network $\pi_{\theta^3}(a_t|s_t)$ in step 3 based on that of step 2, we train it as follows. The agent collects trajectory by interacting with the environment, then updates the policy $\pi_{\theta^3}(a_t|s_t)$ using Equation (1), while the KD divergence with the policy of phase 2 is added $D_{KL}(\pi_{\theta^2}(s_t), \pi_{\theta^3}(s_t))$ as follows:

$$L^{CLIP}(\theta^3) = \hat{\mathbb{E}}\left[\min(r_t(\theta^3)\hat{A}_t, \text{clip}(r_t(\theta^3), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right] + D_{KL}(\pi_{\theta^2}(s_t), \pi_{\theta^3}(s_t)) \quad (2)$$

$D_{KL}(\pi_{\theta^2}(s_t), \pi_{\theta^3}(s_t))$ intuitively leads to deep exploration and expects higher rewards in the future rather than immediate rewards.

Existing RL-based ABR algorithms have environmental dependence, therefore different strategies are required to cope with various environments. The offline phase of multiphase RL allows ABRaider *to have multiple strategies suitable for different environments (i.e., generalization). It paves the way to provide generalized performance to users in the early stages of the online phase.*

C. ONLINE PHASE

We have a generalized ABR algorithm that reinforces the strengths of the existing algorithms generated in the offline phase. Although significant QoE gains have been achieved in the offline phase with multiple strategies (details in Section VI-B), there is still room for improvement. The simulator and datasets used in the offline phase cannot completely cover the real-world diversity. ABR algorithm needs to be specialized in each client's environment, otherwise, it gives some users poor QoE (i.e., starvation effect). In particular, we have explained in Section III-A that a single model with limited capacity is difficult to generalize in various environments. We now describe the online phase leveraging federated learning, which allows *ABRaider* to be trained for a specific client. It consists of local model updates on the client-side and global model updates on the server-side. In addition, we propose the RobustFedAvg algorithm to enhance data efficiency and prevent data poisoning attacks in federated learning.

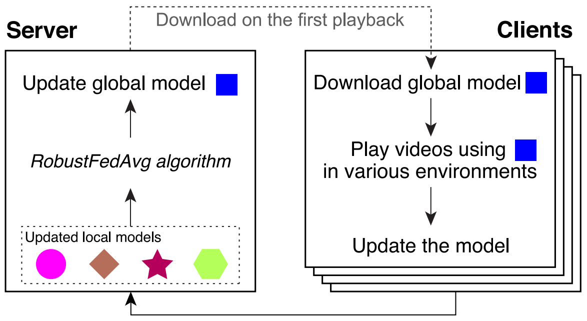


FIGURE 7. Online training process.

1) CLIENT

We explain each component in the online phase and how they operate. An overview of the online training is depicted in Figure 7. When the online phase starts, the client downloads *ABRaider*'s global model $\pi_{\theta_G}(a_t|s_t)$ (at this point, the model version number and client ID are tagged). The client plays the videos with the bitrate determined by *ABRaider*, and obtains the trajectories $\{s_0, a_0, r_1, \cdots, s_{T-1}, a_{T-1}, r_T, s_T\}$. When the number of time steps in the trajectories exceeds the threshold Θ_t (empirically set to 128), the clients update the model using Equation (1). Finally, the updated local model $\pi_{\theta_L}(a_t|s_t)$ is sent to the server with the client ID. By repeating this process, the client contributes to the improvement of *ABRaider* while specializing it to specific environments in a crowdsourcing manner. Note that to ensure specialization of the user's environment, the client downloads the global model when it does not have a local model or when it is first played. Otherwise, the local model trained for the user's environment will be replaced by a global model generalized to all users, making specialization difficult.

2) SERVER

Upon receiving the global model request, the server responds with the latest global model along with the newly generated client ID. When the server receives the local model from the client, it stores the model with the client ID. When the number of local models reaches the threshold Θ_m (set to 64), the global model is updated using the RobustFedAvg algorithm, a variant of the FedAvg algorithm [28]. The model version number and client ID are used to control global model version and to identify each client, respectively.

3) ROBUSTFEDAVG ALGORITHM

The FedAvg algorithm does not validate the locally updated models, therefore it has the following shortcomings: (i) since it simply averages the models obtained from clients, the global model is not always updated with better performance (i.e., high variance), (ii) it is vulnerable to the data poisoning attack. We propose a RobustFedAvg algorithm, which extends the FedAvg algorithm in order to increase the data efficiency and protect against the data poisoning attack. The pseudo-code is presented in Algorithm 1. The round starts, when the number of local models held by the server reaches Θ_m (line 2). The number of clients per group is determined by the fraction of the clients C (set to 0.1) (line 3). In each group, m number of clients are randomly selected (line 6). The server obtains the local models of the selected clients using a function *LoadLocalModel* (lines 7-8). The models are averaged and appended to an array *candidate* (lines 9-10). Each model in the *candidate* array is validated and the global model from the previous round is replaced by the best model using a function *PickBestModel* (lines 11-12). To validate the candidate models in *PickBestModel*, we use the simulator in Section V-B and calculate QoE with Equations (3) and (4).

Algorithm 1 RobustFedAvg

```
1:  $K$ clients are indexed by $k$.
2:  for each round $t = 1, 2, \ldots$ do
3:      $m = \max(C \cdot K, 1)$
4:      $candidate = []$
5:      for each group $l = 1, 2, \ldots, \lceil \frac{K}{m} \rceil$ do
6:          $S_t$ = (random set of $m$ clients)
7:          for each client $k \in S_t$ in parallel do
8:              $w_{t+1}^k = LoadLocalModel(k)$
9:          end for
10:         $w_{t+1} = \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$
11:         $candidate$.append($w_{t+1}$)
12:     end for
13:     $best = PickBestModel$(candidate)
14:     $GlobalUpdate(best)$
15: end for
```

4) DEPLOYMENT

To expedite the online phase through crowdsourcing, we hosted *ABRaider*'s neural network model on a publicly available server [29]. It gives anyone an opportunity to improve *ABRaider* by accessing a web page and playing videos. We prepared several example videos on the media server and implemented a video player using dash.js [13] on the webserver. An HTTP interface is configured for exchanging both the trajectories and updated models with the clients.

It is almost impossible to make the ABR algorithm guarantee high QoE to all users with a single model in the real world. Therefore, we train ABRaider *online in a distributed manner at the client level, specializing in users' individual environments. Furthermore, each local model is periodically reflected in the global model, and the baseline of the online phase is continuously developed.*

D. NEURAL NETWORK AND QoE METRIC

Here, we describe the details of ABRaider's neural network and QoE metric used for training and evaluation. We first enumerate the inputs of the neural network and then describe each module and output of the neural network. Finally, we introduce $QoE_{ABRaider}$, a variant of QoE_{lin} that consider video segment length.

1) NEURAL NETWORK

We design a neural network dedicated to the ABR algorithm. *ABRaider's* neural network is based on the actor-critic architecture [30] and consists of a representation layer, an actor head, and a critic head as depicted in Figure 8. The representation layer is a module that encodes the observations received from a video streaming environment and consists of a condition encoder and a property encoder. The output of each encoder is concatenated and fed to the actor head and the critic head, respectively. The actor head outputs a probability for each bitrate $\pi_\theta(a_t, s_t)$, and the critic head predicts the reward $v^{\pi_\theta}(s_t) = r_t$ (more precisely, a discounted sum of rewards). The actor head and critic head consist of two fully connected layers with residual connections [31] and a single output layer. This segmentation of the representation layer has the following advantages. (i) It is easy to visualize inputs passing through the representation layer, therefore we can confirm the extracted features and abstractions. (ii) Since transfer learning can be applied to each encoder in the representation layer, it allows flexible domain expansion and reuse in different domains.

We define the inputs of neural network $s_t = (\vec{c}_t, \vec{r}_j, b_t, l, x_t, h_t, v)$, where $\vec{c}_t$ is the estimated throughputs for the past 20 video segments, $\vec{r}_j$ is the available bitrates at index j, b_t is the buffer occupancy at time step t, l is the segment length, x_t is the number of segments left, h_t is the last selected bitrate, and v is streaming type (i.e., VoD or live streaming). Note that we (i) separate the segment length from the available bitrate, not the segment size, and (ii) consider the streaming type as an input observation. (i) helps a neural network learn various strategies depending on the segment length and the available bitrates. (ii) allows a neural network to have a different strategy according to the streaming type. For example, the ABR algorithm must be more conservative in live streaming than VoD because the maximum playback buffer is limited.

2) QOE METRIC

The authors of MPC have well defined QoE_{mpc} to quantify the QoE perceived by the users. Despite the author's intent, QoE_{mpc} tends to be calculated differently with respect to environment configuration (e.g., segment length). They assumed

TABLE 3. The public datasets we used.

Dataset	Network	Throughput (Mbps)	Description
FCC [32]	Broadband	mean: 2.89, std: 1.37	Raw data collected from 7 ISPs in the U.S. We use the throughput traces of accessing Youtube in web browsing category.
UCC [33]	LTE	mean: 14.47, std: 15.57	We use throughput traces of streaming video with 5 mobility patterns: static, pedestrian, car, tram, and train.
UCC5G [34]	5G/LTE	mean: 9.24, std: 11.10	Contains 2 mobility patterns (static and car), and 2 applications (video and file downloading).
Belgium [35]	LTE	mean: 3.18, std: 1.96	Consists of 40 traces of 6 transportation types; foot, bicycle, bus, tram, train, and car
Norway [36]	3G/HSDPA	mean: 1.60, std: 0.80	Contains 30 minutes of video streaming using transportation (bus, ferry, tram, train, and car).
Oboe-trace [10]	Wi-Fi/3G/LTE	mean: 2.77, std: 1.50	Dataset collected while streaming video.

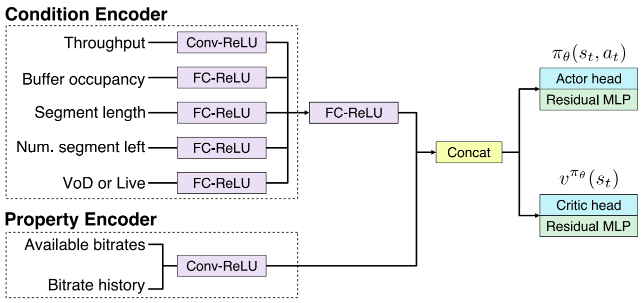


FIGURE 8. ***ABRaider's*** **neural network architecture.**

a segment length of 4 seconds, however this assumption leads to the problem of having a different penalty ratio for video with a segment length other than 4 seconds. To mitigate this discrepancy, we introduce $QoE_{ABRaider}$ (a variant of QoE_{mpc}) taking into account the segment length,

$$QoE_{ABRaider} = \sum_{n=1}^{N} \rho L q(R_n) - \mu \sum_{n=1}^{N} T_n - \sum_{n=1}^{N-1} |q(R_{n+1}) - q(R_n)| \quad (3)$$

where R_n is the quality (i.e., bitrate) of a video segment, L and ρ is segment length and quality weight, respectively. The first term in Equation (3) is the aggregation of the video bitrate which corresponds to the quality. In the second term T_n is the rebuffering time, thus it corresponds to a rebuffering penalty that negatively affects QoE. We set ρ to 0.25 in order to make the scale of QoE_{mpc} and $QoE_{ABRaider}$ the same. Furthermore, we consider the following $q(\cdot)$ which has a marginal improvement for higher bitrates and sensitive to rebuffering

$$q(R_n) = log(R_n/R_{min}) \quad (4)$$

The third term penalizes the quality changes. The weight μ for rebuffering in Equation (3) is set to 2.66 as used in [7]. We use each QoE component in Section VI-B.

VI. EVALUATION

We evaluate the performance of *ABRaider* by answering the following questions.

- How does *ABRaider*'s offline phase perform compared to the baseline algorithms?
- Does *ABRaider* effectively absorb the strengths of rule-based algorithms? As a result, how does generalization ability compare to traditional RL approaches?
- How does *ABRaider*'s online phase effectively specialize in a given environment?

A. EXPERIMENTAL SETUP

1) NETWORK TRACES

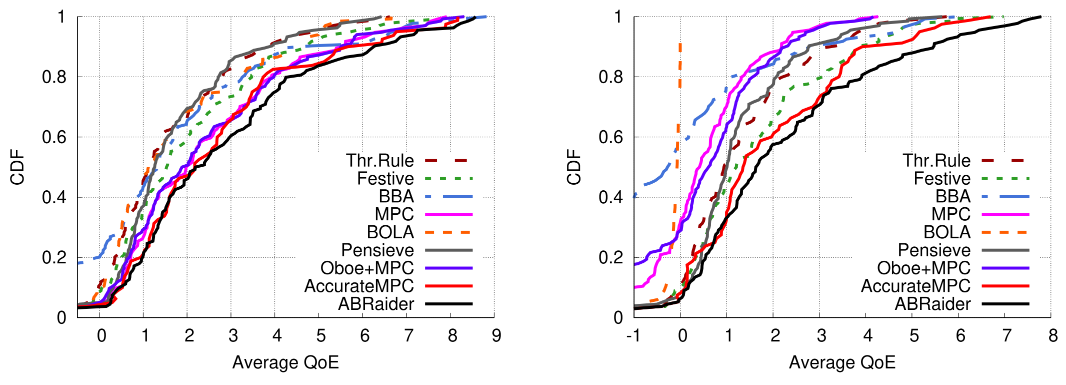
We use 6 public datasets collected from real networks (details in Table 3) for training and evaluation. We divide each dataset by a ratio of 9:1 for training and test sets, respectively. In addition, synthetic traces are generated using two network characteristics, bandwidth and variations. For this, we use the Markovian model; each state represents the average bandwidth and the bandwidth variations trigger state transition. We set the average bandwidth to 3, 6, and 15 Mbps, and the bandwidth variation to stable, medium and extreme. There are 9 categories (bandwidth and variation pairs) and each category has 200 traces of 10,000 seconds long. The total network trace is about 266 hours.

2) VIDEOS

Followings are used for training and evaluation; (i) *Envivio-dash3*, bitrate set is {0.3, 0.8, 1.2, 1.6, 2.6, 4.3} Mbps, and segment length is 4 seconds. (ii) *Big Buck Bunny (BBB)*, bitrate set is {0.2, 0.5, 0.8, 1.0, 1.3, 1.9, 3.1, 5.0, 9.9, 14.9} Mbps, and segment length is 4 seconds. (iii) *High Quality Aerial (HQA) [37]*, bitrate set is {1.1, 2.3, 3.9, 5.8, 7.9, 12.5, 18.0, 24.0, 38.4, 65.6} Mbps, and segment length is 8 seconds. (iv) *Synthetic*, we use video segment length and bitrate range to generate synthetic videos. Similarly, we set 3 levels for each and combine them to create 9 categories. Uniform random noise is added to the bitrate to mimic variable bitrate. Each category contains 100 videos. Note that the aforementioned videos are set to a segment length of 0.5 seconds in live streaming, which is low-latency CMAF configurations of Wowza [38].

3) BASELINE ALGORITHMS

We compare *ABRaider* to the following state-of-the-art algorithms.



(a) VoD

(b) Live streaming

FIGURE 9. On average, ***ABRaider*** provides higher QoE than others in both VoD and live streaming.

- Rate-based algorithms only use the throughput of past segments. *ThroughputRule* and *Festive* apply moving average and harmonic mean to past throughputs to predict the next bandwidth, respectively.
- Buffer-based algorithms determine the bitrate based on playback buffer occupancy. *BBA* determines bitrate in proportion to buffer occupancy, and *BOLA* uses Lyapunov optimization.
- ML-based algorithms are data-driven approaches instead of fixed rules. *Pensive* uses reinforcement learning to directly improve QoE, and *Fugu* trains a Transmission Time Predictor (TTP) module to predict the transmission time of future segments, which works with MPC. However, we use Fugu's upper bound performance, *AccurateMPC*, due to the implementation issues (Since Fugu is a server-side algorithm and uses additional internal TCP statistics, we compare *ABRaider* to AccurateMPC instead of directly comparing to Fugu).

4) IMPLEMENTATION AND TRAINING DETAILS

Our implementation consists of a media server and a client. We use MPEG-DASH for *ABRaider's* client. The server includes an apache server for handling request and response. For fair comparison, we use the FCC, Belgium and Norway datasets in Table 3 to train *ABRaider* in the offline phase, the same as in [9]. The video training set includes all videos except HQA. We use the simulator for offline phase training of *ABRaider*, and its implementation follows [9]. We train *ABRaider* with Adam optimizer [39] for 300k steps with a learning rate of $1e^{-4}$. We train in PyTorch during the offline phase, convert the trained model to the TensorFlow model, and then use TensorFlow.js for the online phase.

B. OVERALL PERFORMANCE

We configure 135 distinct environments using network traces and videos. In each case, a total of 40 playbacks are performed and the average QoE is calculated. The CDFs of average QoE for *ABRaider* and ABR algorithms are presented in Figure 9. We confirm that *ABRaider* provides the best performance in both VoD and live streaming. Specifically, in VoD, *ABRaider*'s average QoE is 19.9% higher than Oboe+MPC (2nd best) as shown in Figure 9a. Moreover, *ABRaider* is superior to AccurateMPC which takes advantage of perfect bandwidth prediction. To quantify the *ABRaider's* QoE gains, we present detailed values of each QoE component; quality, rebuffering and smoothness penalties, in Figure 10. It can be seen that *ABRaider* is not the best in each component, but it strikes a good balance. *ABRaider* uses the most suitable strategy for the environment through multi-phase RL, and hence achieves QoE improvement.

In live streaming Figure 9b, *ABRaider* achieves significant QoE improvements, 42.2% higher than Festive (2nd best). BBA, BOLA, MPC and Oboe, which use buffer occupancy as an observation, suffer from the limitation of the playback buffer in live streaming. Rather, rate-based algorithms Festive and ThroughputRule show much better performance except *ABRaider*. When training, *ABRaider* considers streaming type as one of the observations and develops strategies that fit the environment. Figure 10b shows that *ABRaider*'s quality is significantly higher than others, but with similar rebuffering and quality change penalties. *ABRaider* has a slightly higher quality changes (i.e., smoothness penalty) in live streaming compared to VoD. We assume that in case of limited buffer occupancy, *ABRaider* responds quickly to the estimated throughput to avoid rebuffering. Nevertheless, the improvement in quality far outweighs the penalties and eventually leads to high QoE.

A comparison of the ABR algorithm in live streaming provides an opportunity to confirm several facts. First, live streaming is one of the extreme cases among various environments, allowing us to demonstrate the environment-adaptive ability of the ABR algorithm. Unlike VoD, the maximum buffer size is very small and video segments are forced to be less than 1 seconds. At this time, in order to guarantee high QoE, the ABR algorithm adapts to these environments and requires different strategies than in VoD. Second, the sensitivity of the ABR algorithm strategy can be demonstrated. Unlike in VoD, in live streaming, only slightly aggressive or

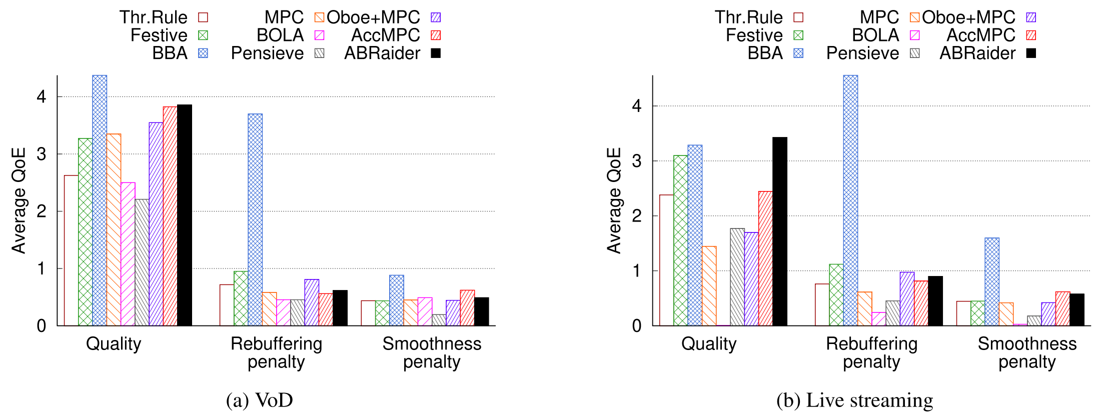


FIGURE 10. ***ABRaider*** **strikes a good balance, and hence it improves QoE.**

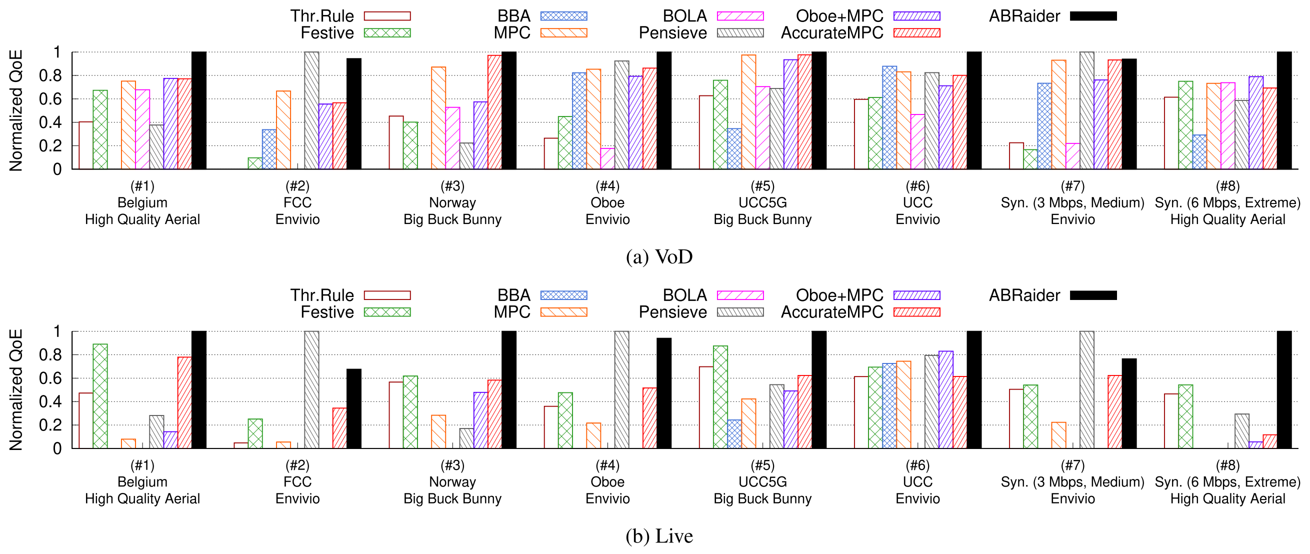


FIGURE 11. ***ABRaider*** **constantly provides high QoE in most configurations, except #2 and #7 in live streaming.**

conservative decisions have a significant impact on QoE due to the limited buffer size. For example, even if the current buffer is full, even some aggressive decisions can lead to buffer stall. Thus, the comparison of performance in live streaming can show whether the ABR algorithm is capable of sensitive adaptations.

C. MICROSCOPIC VIEW

We confirmed the *ABRaider's* performance gains in various environments. Now we look into detailed comparisons in 8 environments, consisting of 6 traces described in Table 3 and 2 synthetic traces (3 Mbps with medium variation and 6 Mbps with extreme variation). For evaluation, in each case we assign one of three videos: Envivio, BBB and HQA. We use the normalized QoE (divided by max. QoE) because the maximum QoE varies from the environment (depending on bandwidth, streaming type, video and etc.).

1) SUBTLE ADAPTATION

Normalized QoEs for all schemes are presented in Figure 11. *ABRaider* is superior to others in most cases, and especially stands out in environments #1, #3, #6 and #8. In #1 and #3, the network bandwidth is considerably lower than the video bitrate range, and the bandwidth fluctuations are high in #6 and #8. In these environments, *ABRaider's* QoE improvement over the 2nd best algorithm is 32.9% on average. This result reveals that *ABRaider*'s QoE gain is large in harsh environments where it is difficult to ensure high QoE. Sensitive adaptation is required in environments where the video bitrate is relatively higher than the available bandwidth or the bandwidth is highly variable.

2) ABRAIDER VS. RL-BASED

The performance of the RL-based algorithm is tightly coupled with the training environment. We can see from Figure 11a, Pensieve provides high QoE in #2, #4, and #7 which are similar to Pensieve's training configurations, FCC and Norway traces and Envivio video. It is no wonder Pensieve outperforms *ABRaider* in #2 and #7. Unlike *ABRaider*, Pensieve is fine-tuned in a given environment therefore it has a limited policy that provides optimal performance in

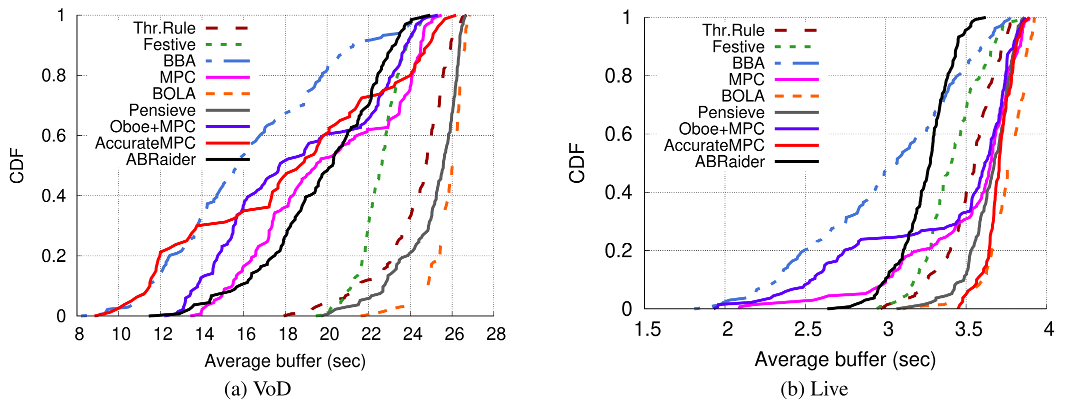


FIGURE 12. **Average buffer occupancy.**

such environment. *ABRaider* outperforms Pensieve in most environments due to the variety of strategies gained through multi-level RL.

3) ABRAIDER VS. SL-BASED

We compare *ABRaider* with AccurateMPC instead of Fugu, a state-of-the-art SL-based ABR algorithm. SL-based approaches generate data-driven bandwidth predictors and work with rule-based algorithms such as MPC. As a result, they provide more accurate observations in MPC, however it is difficult to have multiple strategies because the rule-based algorithm determines the bitrate. In Figure 11a, AccurateMPC is slightly better than MPC in most environments but does not provide good QoE for #8. This is because MPC's ABR policy determining the bitrate is not sufficient, although it comes with completely accurate bandwidth predictions.

4) ABRAIDER VS. RULE-BASED

We verify the effectiveness of the offline phase compared with the rule-based algorithms, ThroughputRule, Festive, BBA, MPC and BOLA as shown in Figure 11. *ABRaider* properly employs and enhances the strengths of these algorithms in order to suit a variety of environments. In addition, offline phase RL can easily merge new adaptation algorithm into *ABRaider* and improves it. This is the core concept of the offline phase.

5) WHY DOES ABRAIDER WORK WELL?

To analyze *ABRaider*'s behavior, we compare the buffer occupancy. Figure 12 shows the buffer occupancy collected from the entire test environments. As shown in Figure 12a, *ABRaider's* buffer occupancy is evenly distributed between 12 and 22 seconds. We can infer that *ABRaider* takes different strategies depending on the environment. On the other hand, other ABR algorithms tend to maintain a specific buffer level. ThroughputRule, Festive, BOLA, and Pensieve maintain a high buffer occupancy (over 20 seconds) regardless of the environment. Since MPC, Oboe+MPC, and AccurateMPC have two strategies, aggressive and conservative adaptation, buffer occupancy can be clustered into two that eventually fail to respond adequately to the environment.

Given that maximum buffer occupancy is limited in live streaming, *ABRaider* maintains between 3 and 3.5 seconds as shown in Figure 12b. This allows *ABRaider* to deliver high QoE while avoiding sudden drops in bandwidth. On the contrary, BOLA and Pensieve are both conservative, and MPC and Oboe+MPC are too aggressive, because they do not differentiate between live streaming and VoD.

D. EXPERIMENTS IN REAL-WORLD

We have seen *ABRaider's* QoE gains in various trace-driven environments. Now we turn our attention to experiments in practice. Specifically, unlike previous experiments, we implement a DASH client on a web page and throttle its bandwidth with network traces. For comparison, we use DYNAMIC [18], [19], MPC and Oboe+MPC. DYNAMIC is the default ABR algorithm implemented in dash.js, which is a combination of ThroughputRule and BOLA. For repeatable and fair comparisons, we first collect 4 types of network traces, campus Wi-Fi, public Wi-Fi in coffee shop, 5G-static and 5G-mobile (car), which are the most representative networks in daily life. The campus Wi-Fi traces contain sharp drop in bandwidth due to hand-off of mesh Wi-Fi (average bandwidth $\sim$20 Mbps). The average bandwidth of coffee shop trace is about 15 Mbps, but due to the sharing nature of public Wi-Fi, it varies significantly in the range of 1-30 Mbps. For 5G traces, we randomly select each from the static and car categories of the UCC5G dataset. Combining 4 network traces, 2 videos (BBB and HQA) and streaming types (VoD or live), 12 experimental environments are configured. Then, we emulate 4 algorithms while throttling the client's bandwidth using network traces.

1) PERFORMANCE COMPARISON

In Figure 13, we present the experiment results of 4 algorithms in 12 environments. The y-axis represents the normalized QoE. Again, *ABRaider* outperforms others in most cases. Especially, *ABRaider* achieves significant

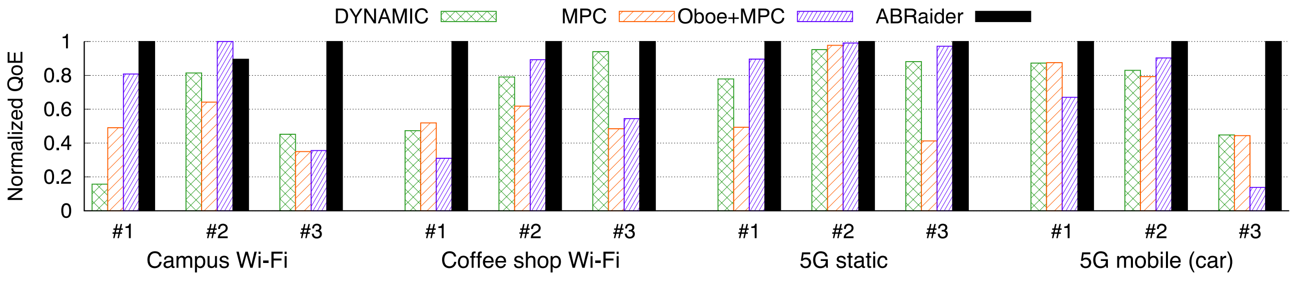


FIGURE 13. Experiment results (#1: HQA in VoD, #2: BBB in VoD, #3: BBB in live streaming).

TABLE 4. Computation time (msec).

Thr. Rule	Festive	BBA	MPC
0.62	0.09	0.00	18.59

BOLA	Pensieve	Oboe+MPC	ABRadier
0.17	1.11	117.17	2.58

performance improvements in #1 and #3 of both campus and coffee shop Wi-Fi environments where sensitive adaptation is required due to the characteristics of the networks. In a typical configuration (i.e., #2 of all networks), the performance of the comparing algorithms is close to *ABRaider*.

We highlight that *ABRaider* has almost no performance discrepancy between real-world and trace-driven results, although it is trained using simulator and the traces. Its real-world experiment results are very similar to simulation results in Figure 11. We can conclude that offline phase training paves the way to the performance guarantee in the real world.

2) COMPUTING OVERHEAD

Once the ABR algorithm receives observations, the computation time is a factor that determines the time until the segment download starts. Thus, computation time affects user engagement in practice, and eventually QoE [40]. In this sense, it is necessary to consider the computation time in ABR algorithm. The estimated throughput $\hat{C}_k$ considering computation time can be written $\hat{C} = \frac{LR}{t_e - t_s + d}$ where, t_s and t_e are the download start time and end time, respectively. L is the segment length and R is the bitrate of the segment. d is the delay until the segment download begins, i.e., sum of latency and computation time. Intuitively, if smaller segments are downloaded in a short time, d attenuates the estimated throughput.

Table 4 summarizes the computation time measured for all algorithms. *ABRaider's* 2.58 msec is considerably shorter than those of MPC and Oboe+MPC. Despite having multiple adaptation strategies, ABRaider maintains negligible inference time. MPC maximizes the QoE metric over a look-ahead window of 5 future chunks [7], resulting in a longer computation time. When the computation time is long, the algorithm tends to request low bitrates to fill the buffer quickly, which in turn leads to low QoE. This phenomena has a negative impact on live streaming. Considering live streaming (#3), DYNAMIC yields QoE similar to Oboe+MPC on Campus Wi-Fi and 5G static, however DYNAMIC excels on Coffee shop Wi-Fi and 5G mobile with large bandwidth fluctuations.

E. *ABRaider* VS. SUPER-RESOLUTION

The super-resolution is a very useful technology for improving the user's QoE when bandwidth is scarce because it directly enhances low-quality segments to high quality using DNN (deep neural network). Here, we compare the performance of *ABRaider* and the super-resolution model, NAS-MSDR [41]. In NAS-MSDR, the server stores both the video segments and DNN chunks, and the client downloads either segments or DNN chunks according to its adaptation algorithm. The more DNN chunks are received, the higher super-resolution gain can is achieved, therefore full DNN provides the best performance. In our NAS implementation, we use MPC for video adaptation and assume that the client stores the entire DNN model before video playback to obtain the best performance. We create 5 environments with different bandwidth ranges, FCC (2 and 6 Mbps, stable and variable) and coffee shop Wi-Fi (used in Section VI-D), and use News (bitrate set is {0.4, 0.8, 1.2, 2.4, 4.8 Mbps}) [42], the same video used in NAS-MSDR.

Figure 14 shows the normalized QoE. Compared to MPC, NAS+MPC yields higher QoE in case of low bandwidth, FCC-2 Mbps (both stable and variable). On the contrary, *ABRaider* is slightly better in network FCC-6 Mbps. The reason is that as the segment quality increases, the gain of super-resolution diminishes. In particular, NAS+MPC has less gain in coffee shop Wi-Fi with the bandwidth configured up to 20 Mbps. Although *ABRaider* does not directly improve video quality, it can compete with or exceed NAS+MPC in environment where the bandwidth is not too low.

The super-resolution gain of NAS+MPC decreases when the bandwidth fluctuates heavily. NAS+MPC can only enhance the unplayed segments in the playback buffer, therefore the higher the buffer occupancy, the more opportunities for super-resolution. However, bandwidth dynamics hinders maintaining buffer level high, reducing the chance of super-resolution. For instance, the QoE gap between NAS+MPC and *ABRaider* in networks FCC-variable (2 and 6 Mbps) is smaller than FCC-stable (2 and 6 Mbps).

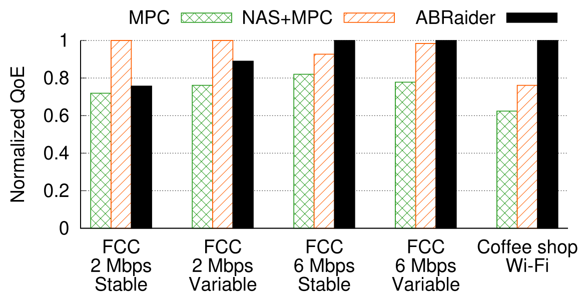


FIGURE 14. ***ABRaider* provides higher QoE than NAS+MPC when the network bandwidth ≥ 6 Mbps.**

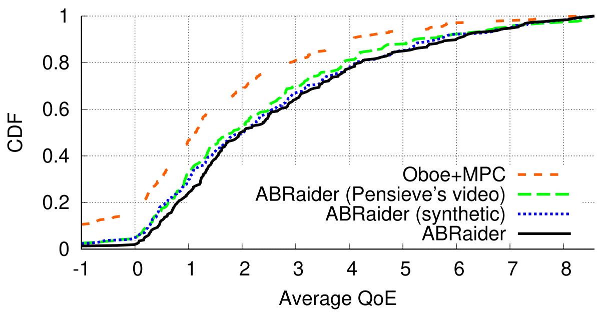


FIGURE 15. ***ABRaider's* variants have similar performance, no dependence on network traces and videos.**

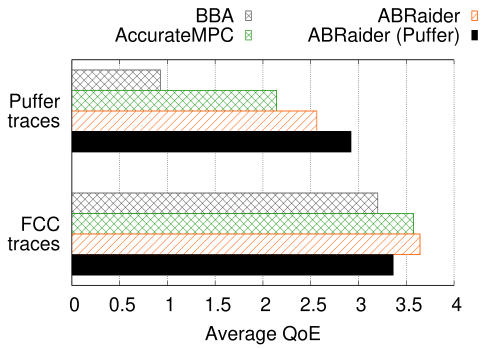


FIGURE 16. **Online phase provides specialized performance in a specific environment.**

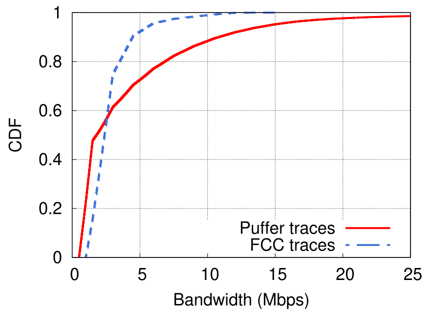


FIGURE 17. **Bandwidth distribution of puffer and FCC traces.**

F. DEPENDENCE ON TRACES AND VIDEOS

As we seen in Section VI-C, RL-based algorithm could be highly dependent on the training environment. In particular, we can see in Figure 11, Pensieve provides high QoE in FCC-Envivio (#2), whereas fairly low performance in Norway-BBB (#3). This shows that trained with FCC and Norway traces and Envivio video, Pensieve is highly reliant on videos and bandwidth range. To evaluate the dependence of *ABRaider*'s training scheme on network traces and videos, we create several variants of *ABRaider* using different network traces and videos. We generate 3 variants using real network traces, synthetic traces and Pensieve's video generator, respectively. Network traces collected in the real-world have The network traces collected in the real-world have an average bandwidth of 3.6 Mbps and are configured up to 20 Mbps. Synthetic network traces consist of an average of 3, 6, and 15 Mbps with different variations as described in Section VI-A (maximum of 30 Mbps). Pensieve's video generator has a narrow range of bitrates (up to 6 Mbps) and lacks of consideration for VBR.

Figure 15 presents a performance comparison between the three variants tested in 135 environments. They show similar QoE even though they are trained on different network traces and videos. Here are some inferences from the results. First, the offline phase is independent of the network bandwidth range, and only needs various network characteristics such as high fluctuations and sudden drops in bandwidth. Because of this *ABRaider* trained with synthetic traces that do not contain as diverse characteristics as real traces is slightly inferior to others. Second, *ABRaider*'s training scheme has almost no dependence on videos. Although the maximum bitrate of Pensieve's generator is 6 Mbps, the offline phase of multi-phase RL allow *ABRaider* to learn strategies regardless of the specific video characteristics.

G. INDIVIDUAL SPECIALIZATION

We confirmed that *ABRaider* trained only in the offline phase already achieved significant performance gains. Further performance improvements are expected through the online phase using crowdsourcing. Given the difficulties of collecting many trajectories to obtain meaningful results in the online phase, we validate the performance of the online phase using public datasets (i.e., trace-driven approach). The client performs local updates using trajectories generated by playing videos with the traces provided by the video streaming website, Puffer [12]. We extract network traces from August 30 to October 16, 2019 from Puffer. The client trains the global model in specific environments made up of Puffer traces for about 15 days (360 hours), which allows *ABRaider* to specialize in Puffer traces. We refer to this model as *ABRaider (Puffer)*. We evaluate the performance of the online phase by comparing *ABRaider (Puffer)* with BBA, AccurateMPC, and *ABRaider*.

Figure 16 shows the average QoE for each algorithm in environments using Puffer and FCC traces. The distribution of each trace is shown in Figure 17, indicating that the two environments have different characteristics. *ABRaider (Puffer)* outperforms in the environments with Puffer traces.

TABLE 5. Average QoE for three *ABRaiders* with and without data poisoning.

Scheme (time steps)	Poisoning	No poisoning
Offline (322k)	–	2.462
Online+FedAvg (580k)	1.685	2.347
Online+RobustFedAvg (417k)	2.323	2.424

It is intensely trained in particular environments and has a large improvement beyond the *ABRaider* (i.e., offline phase). In FCC traces that have never experienced, *ABRaider (Puffer)* is inferior to *ABRaider* and AccurateMPC. In summary, the online phase allows *ABRaider* to evolve into an ABR algorithm that takes individual users into account by intensive training on specific environments.

H. VALIDATION OF ONLINE PHASE

In this section, we evaluate the data efficiency of the RobustFedAvg algorithm and robustness against data poisoning attacks. We create 5 *ABRaiders* with different configurations: training phase (offline and online), global model update algorithm (FedAvg and RobustFedAvg) and data poisoning attack. To mimic data poisoning attacks, we randomly select 5% of clients per update (round) and damage their trajectories.

Table 5 shows the average QoE obtained in 135 environments. In the absence of the data poisoning attack, all three *ABRaiders* perform similarly, so it can be inferred there is no difference between the offline phase and the online phase. Note that the data efficient RobustFedAvg requires fewer time steps (417k) to converge than the FedAvg algorithm (580k). The RobustFedAvg algorithm is more resistant to data poisoning attacks than the FedAvg. In data poisoning attacks, *ABRaider* with the RobustFedAvg provides much higher QoE than the FedAvg.

VII. FUTURE WORK

For future work, we extend *ABRaider* beyond the current experimental environments to target a public streaming application service. Specifically, real services differ in two aspects: (i) highly dynamic environments, and (ii) heterogeneous computing power of mobile devices.

First, in fact, when users use a video streaming service, they usually use a mobile device such as a smartphone. In addition, since a smartphone has multiple network interfaces such as cellular or Wi-Fi, network characteristics frequently change within a short period of time during video playback by switching network interfaces as well as by changes in RSSI. Our experimental environment includes various environments, but does not reflect these characteristics. To alleviate this problems, we need to make the horizon (i.e., batch size) smaller, but this can lead to unstable learning. Therefore, to address this limitation, we envision employing some training techniques such as few-shot learning or meta-learning.

Second, as mentioned in Section III-B and VI-D, *ABRaider* is a lightweight model and recent mobile devices are equipped with neural engine GPUs. However, since existing mobile devices have different computing powers, the same decision (i.e., bitrate) can lead to different results. For example, on low-performance devices, it takes longer to observe observations and start downloading segments (i.e., inference time), resulting in rebuffering events or lowering buffer occupancy. To alleviate this, we need techniques that can adjust the network capacity according to the device's performance while minimizing the accuracy loss. We consider techniques such as knowledge distillation and scalable neural networks. At the end of this, by adjusting the complexity of the neural network according to the computing power, it is possible to minimize the policy discrepancy between devices with various performances.

VIII. CONCLUSION

Video streaming is one of the killer applications these days. Many ABR algorithms are proposed to provide users with satisfactory video streaming, but it is difficult to guarantee high QoE in various environments. In this work, we present the design, implementation and evaluation of *ABRaider* – a multi-phase RL-based ABR solution for video streaming. *ABRaider* aims to provide high QoE to all users in various environments in the real world. The offline phase leverages the strengths of the ABR algorithms and develops multiple strategies according to the environment (i.e., generalization), as well as easily can merge new adaptation algorithms. The online phase allows for specialization in individual client environments by utilizing its computational power to continue training. We demonstrate its superiority over conventional approaches through experimentation in broad set of networks and videos.

REFERENCES

[1] *Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2015–2020*. Accessed: May 15, 2022. [Online]. Available: https://www.cisco.com/c/dam/m/en_in/innovation/enterprise/assets/mobile-white-paper-c11-520862.pdf

[2] *Mobile Internet Phenomena Report*. Accessed: May 15, 2022. [Online]. Available: https://www.sandvine.com/phenomena/

[3] T. Stockhammer, "Dynamic adaptive streaming over HTTP—Standards and design principles," in *Proc. 2nd ACM Conf. Multimedia Syst.*, 2011, pp. 133–144.

[4] *Optimal Adaptive Streaming Formats MPEG-DASH & HLS Segment Length*. Accessed: May 15, 2022. [Online]. Available: https://bitmovin.com/mpeg-dash-hls-segment-length/

[5] J. Jiang, V. Sekar, and H. Zhang, "Improving fairness, efficiency, and stability in HTTP-based adaptive video streaming with FESTIVE," in *Proc. 8th Int. Conf. Emerg. Netw. Exp. Technol. (CoNEXT)*, 2012, pp. 97–108.

[6] 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," in *Proc. ACM Conf. SIGCOMM*, Aug. 2014, pp. 187–198.

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

[8] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman, "BOLA: Near-optimal bitrate adaptation for online videos," in *Proc. 35th Annu. IEEE Int. Conf. Comput. Commun. (INFOCOM)*, Apr. 2016, pp. 1–9.

[9] H. Mao, R. Netravali, and M. Alizadeh, "Neural adaptive video streaming with pensieve," in *Proc. ACM Conf. SIGCOMM*, 2017, pp. 197–210.

[10] Z. Akhtar, Y. S. Nam, R. Govindan, S. Rao, J. Chen, E. Katz-Bassett, B. Ribeiro, J. Zhan, and H. Zhang, "Oboe: Auto-tuning video ABR algorithms to network conditions," in *Proc. ACM Conf. SIGCOMM*, 2018, pp. 44–58.

[11] Y. Sun, X. Yin, J. Jiang, V. Sekar, F. Lin, N. Wang, T. Liu, and B. Sinopoli, "CS2P: Improving video bitrate selection and adaptation with data-driven throughput prediction," in *Proc. ACM Conf. SIGCOMM*, 2016, pp. 272–285.

[12] 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 *Proc. 17th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2020, pp. 495–511.

[13] *DASH Industry Forum*. Accessed: May 15, 2022. [Online]. Available: https://dashif.org/

[14] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman, "BOLA: Near-optimal bitrate adaptation for online videos," *IEEE/ACM Trans. Netw.*, vol. 28, no. 4, pp. 1698–1711, Aug. 2020.

[15] H. Zhang, A. Zhou, J. Lu, R. Ma, Y. Hu, C. Li, X. Zhang, H. Ma, and X. Chen, "OnRL: Improving mobile video telephony via online reinforcement learning," in *Proc. 26th ACM Int. Conf. Mobile Comput. Netw.*, 2020, pp. 1–14.

[16] Z. Li, K. Kamnitsas, and B. Glocker, "Overfitting of neural nets under class imbalance: Analysis and improvements for segmentation," in *Proc. Int. Conf. Med. Image Comput. Comput. Assist. Intervent. (MICCAI)*, 2019, pp. 402–410.

[17] C. Zhang, O. Vinyals, R. Munos, and S. Bengio, "A study on overfitting in deep reinforcement learning," 2018, *arXiv:1804.06893*.

[18] (2018). *ABR Logic*. [Online]. Available: https://github.com/Dash-Industry-Forum/dash.js/wiki/ABR-Logic/

[19] K. Spiteri, R. Sitaraman, and D. Sparacio, "From theory to practice: Improving bitrate adaptation in the DASH reference player," *ACM Trans. Multimedia Comput., Commun., Appl.*, vol. 15, no. 2s, pp. 1–29, Apr. 2019.

[20] (2021). *Processors Specification*. [Online]. Available: https://gadgetversus.com/processor/

[21] *Qualcomm—Mobile Processor*. Accessed: May 15, 2022. [Online]. Available: https://www.qualcomm.com/products/mobile-processors

[22] *Market Research: The State of Online Video 2019*. Accessed: May 15, 2022. [Online]. Available: https://www.limelight.com/resources/white-paper/state-of-online-video-2019/

[23] C.-A. Cheng, A. Kolobov, and A. Agarwal, "Policy improvement via imitation of multiple oracles," 2020, *arXiv:2007.00795*.

[24] O. Vinyals *et al.*, "Grandmaster level in StarCraft II using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, Oct. 2019.

[25] S. Reddy, A. D. Dragan, and S. Levine, "SQIL: Imitation learning via reinforcement learning with sparse rewards," 2019, *arXiv:1905.11108*.

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

[27] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," 2015, *arXiv:1506.02438*.

[28] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Y. Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. 20th Int. Conf. Artif. Intell. Statist.*, 2017, pp. 1273–1282.

[29] *Public Website of ABRaider*. Accessed: May 15, 2022. [Online]. Available: http://abraider.site/

[30] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," in *Proc. Adv. Neural Inf. Process. Syst.*, 2000, pp. 1008–1014.

[31] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[32] *Raw Data—Measuring Broadband America—Eighth Report*. Accessed: May 15, 2022. [Online]. Available: https://www.fcc.gov/reports-research/reports/measuring-broadband-america/raw-data-measuring-broadband-america-eighth/

[33] D. Raca, J. J. Quinlan, A. H. Zahran, and C. J. Sreenan, "Beyond throughput: A 4G LTE dataset with channel and context metrics," in *Proc. 9th ACM Multimedia Syst. Conf.*, 2018, pp. 460–465.

[34] D. Raca, D. Leahy, C. J. Sreenan, and J. J. Quinlan, "Beyond throughput, the next generation: A 5G dataset with channel and context metrics," in *Proc. 11th ACM Multimedia Syst. Conf.*, 2020, pp. 303–308.

[35] J. Van der Hooft, S. Petrangeli, T. Wauters, R. Huysegems, P. R. Alface, T. Bostoen, and F. De Turck, "HTTP/2-based adaptive streaming of HEVC video over 4G/LTE networks," *IEEE Commun. Lett.*, vol. 20, no. 11, pp. 2177–2180, Aug. 2016.

[36] H. Riiser, P. Vigmostad, C. Griwodz, and P. Halvorsen, "Commute path bandwidth traces from 3G networks: Analysis and applications," in *Proc. 4th ACM Multimedia Syst. Conf.*, 2013, pp. 114–118.

[37] *High Quality Aerial*. Accessed: May 15, 2022. [Online]. Available: https://www.youtube.com/watch?v=RGoPU-OLQHE/

[38] *The Appetite for Live-Streaming is Bigger Than Ever*. Accessed: May 15, 2022. [Online]. Available: https://www.wowza.com/blog/low-latency-cmaf-chunked-transfer-encoding

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

[40] A. Balachandran, V. Sekar, A. Akella, S. Seshan, I. Stoica, and H. Zhang, "Developing a predictive model of quality of experience for internet video," in *Proc. ACM SIGCOMM Conf. SIGCOMM*, Aug. 2013, pp. 339–350.

[41] H. Yeo, Y. Jung, J. Kim, J. Shin, and D. Han, "Neural adaptive content-aware internet video delivery," in *Proc. 13th USENIX Symp. Oper. Syst. Design Implement. (OSDI)*, 2018, pp. 645–661.

[42] *Shooting Survivor Confronts NRA Spokesperson Dana Loesch*. Accessed: May 15, 2022. [Online]. Available: https://www.youtube.com/watch?v=4AtOU0dDXv8/