Learning Tailored Adaptive Bitrate Algorithms to Heterogeneous Network Conditions: a Domain-specific Priors and Meta-Reinforcement Learning Approach

Tianchi Huang , *Student Member, IEEE,* Chao Zhou, Rui-Xiao Zhang , *Student Member, IEEE,* Chenglei Wu, and Lifeng Sun , *Member, IEEE.*

***Abstract*—Internet adaptive video streaming is a typical form of video delivery that leverages adaptive bitrate (ABR) algorithms to provide video services with high quality of experience (QoE) for various users in diverse and unique network conditions. Such heterogeneous network environments, which can be viewed as exogenous input processes, often lead to the unstable performance of ABR algorithms. Unfortunately, learning-based ABR algorithm which generated by state-of-the-art reinforcement learning (RL) technologies achieves *good average performance* but fails to perform well in all kinds of network conditions.**

In this work, considering the video playback process as the Input-driven Markov Decision Process (IMDP), we propose A^{2}BR (Adaptation of ABR), a novel meta-RL ABR approach. A^{2}BR is mainly composed of an online stage and an offline stage. It leverages meta-RL to learn an initial meta-policy with various network conditions at the offline stage and makes decisions in personalized network conditions at the online stage. At the same time, we continually optimize the meta-policy to the tailor-made ABR policy for varying the current network environment within few shots. Moreover, in order to improve the learning efficiency, we fully utilize domain knowledge for implementing a virtual player to replay the previously experienced network.

Using trace-driven experiments on various scenarios including different vehicles, users, network types, and heterogeneous user-preferences, we show that A^{2}BR outperforming recent ABR approaches with rapidly adapting to the personalized QoE metrics and specific network conditions. Testbed experimental results also illustrate the superiority of A^{2}BR in adapting to the unseen environments.

I. Introduction

Due to the rapid development of network services, video streaming now stands for the predominant Internet application, which is up almost 75% all traffic [1], [2]. Especially, adaptive video streaming, such as HLS (HTTP Live Streaming) [3] and DASH [4] has already been the popular form of video delivery [5]. Adaptive bitrate (ABR) algorithms enable Internet adaptive video streaming services to achieve high video quality while avoiding uninterrupted stall event [5] (§II-A). Revisiting the recent success of ABR algorithms, heuristics often make decisions based on network or player status [6], [7], [8]. However, those schemes require a proper setting of configuration parameters [9], [10] for fitting different network distributions. By contrast, learning-based schemes employ several learning technologies, such as reinforcement learning [11], [12], supervised learning [13], [2] and imitation learning [14], [15] to train a neural network (NN) w.r.t the given network traffic distributions, and make a zero-shot inference for unseen networks. In short, existing ABR algorithms, either heuristics or learning-based schemes, seldom configure or tune their parameters automatically and rapidly for varying the current network traffic distribution.

However, in the adaptive video streaming scenario, the system dynamics are uncertain and the future state cannot be accurately predicted. To prove this view, we focus on investigating the impact of ABR algorithms on the distribution of heterogeneous network traffics, where the distribution is usually summarized by bandwidth traces experienced by different users at any time, in any place, and especially, under any network conditions. Through the analysis of the impact on the network distributions of different users, vehicles, and network types, we empirically find that nowadays' Internet network conditions are not only *diverse* but also *unique* (§II-B). For example, the heterogeneity of network conditions for each user is inevitable, since both subjective and objective user behavior have an important impact on the network traffic distribution. Nevertheless, existing ABR algorithms, either heuristics or learning-based, fail to adapt to such *heterogeneous* bandwidth conditions that are significantly different from the offline training (or tuning) network dataset [16].

Motivated by these facts, we model the ABR playback process as Input-driven Markov Decision Process (IMDP), which can express an implicit heterogeneous network environment in an explicit manner (§III-A). We theoretically illustrate that vanilla RL technologies can only generalize a strategy

T. Huang, and L.Sun are with Beijing Key Lab of Networked Multimedia, Department of Computer Science and Technology, Tsinghua University, Beijing, 100084, China. (e-mail: htc19@mails.tsinghua.edu.cn, sunlf@tsinghua.edu.cn)

RX. Zhang, C. Wu, and L.Sun are with BNRist, Department of Computer Science and Technology, Tsinghua University, Beijing ,10084, China. (e-mail: {zhangrx17, wucl18}@mails.tsinghua.edu.cn)

L.Sun is with Key Laboratory of Pervasive Computing (Tsinghua University), Ministry of Education, China

C. Zhou is with Beijing Kuaishou Technology Co., Ltd, Beijing, China. (e-mail: zhouchao@kuaishou.com)

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

that can perform well on *average* rather than *every network condition*. While through in-depth analysis, we find that the most intuitive solution, i.e., reinforced tailored policies *in situ* [2], is also impractical since off-the-shelf model-free RL methods [17] heavily lack sample efficiency, which cannot train a policy within an acceptable time.

Hence, based on the theory of IMDP, we propose A^{2}BR (Adaption of Adaptive Bitrate Algorithm), a novel neural meta-RL ABR system that enables fast adaptation to the specific network conditions (§III-B). A^{2}BR is composed of the offline stage and online stage (§IV). At the offline stage, A^{2}BR trains a meta-model with various real and synthetic network conditions for learning parameter initialization meta-policy, where the policy can provide rapid adaptation for varying heterogeneous networks. To achieve this goal, we implement the training process based on the state-of-the-art gradient-free meta-learning technology [18] and utilize maximum entropy RL methodologies to achieve better exploration (§IV-B). Moreover, at the online stage, the video player, placed on the user side, receives the trained meta-model and picks the bitrates w.r.t the meta-policy and the current specific network status. Upon finishing the video session, the meta-policy is continually updated to the tailor-made policy with the collected trajectories. For improving the learning efficiency, the trajectories are collected not only from the real world but also from the "virtual world". Specifically, the virtual world is motivated by domain principles and constructed by a faithful virtual player and experienced network environments. In addition, we also employ the domain knowledge that uses heuristics to enable safe online RL. Subsequently, the meta-policy will be continually optimized within 20-shot, i.e., watch 20 videos at the online stage (§IV-C).

In the rest of the paper, we conduct several experiments to evaluate A^{2}BR with existing ABR approaches (§V). The case studies contain different types of heterogeneous network conditions and QoE objectives, including different vehicles, users' personalized networks, 4G/5G networks, and varying user preferences for QoE metrics. Using trace-driven simulation and real-world evaluation on various videos, we show:

1) A^{2}BR improves the video quality by up to 12.6% while reducing the stall time by 69.3% to 2.8× compared with previously proposed approaches.
2) In the user-personalized network, A^{2}BR outperforms recent heuristics and learning-based ABRs, with improvements on average QoE of 12%-23%;
3) A^{2}BR maintains high bitrates with low video stall in both 4G and 5G networks, whereas the learning-based approach Pensieve diverges. At the same time, A^{2}BR either matches or exceeds the performance of existing schemes on IT-T Rec P.1203 QoE metric [19]. The average QoE is 10% higher than the closest ABR approach Fugu [2].
4) A^{2}BR with minor modification can hold QoE metrics with different user preferences, further providing 5% improvements on QoE at the online stage.
5) We prove that A^{2}BR still performs well on both emulation and real-world testbed. Ablution studies show that the online stage further improves the average QoE by 6% after learning in specific network conditions within 10-shot, and 8% after 50-shot.

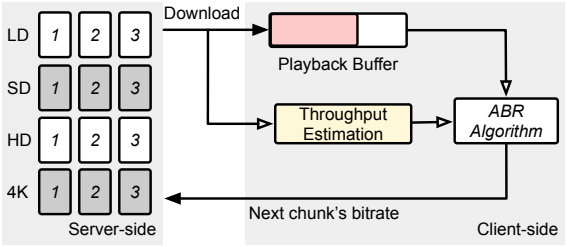


Fig. 1. The typical ABR system overview. The ABR algorithm is usually placed on the client-side.

The contributions of this work are summarized as follows:

- We empirically analyze today's heterogeneous network traffics and propose a two-stage meta-learning scheme for varying specific network conditions.
- We implement A^{2}BR , which is the first meta-learning with domain knowledge approach for adaptive streaming.
- Results on different types of network conditions illustrate that the generated tailor-made ABR policies can well adapt to heterogeneous networks with relatively few-shot.

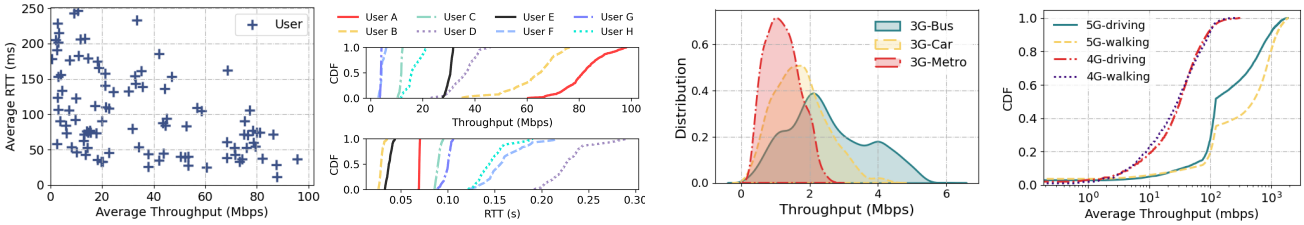
II. Background and Motivation

Our research is started with a fundamental quest: *How will the recent ABRs perform in various network traffic environments?*. To answer this question, first, we briefly introduce the key principle of adaptive video streaming and adaptive bitrate (ABR) algorithms. We then use empirical measurements to elucidate the key limitations of prior solutions.

A. Adaptive Video Streaming

The adaptive bitrate method (ABR) is an algorithm that dynamically selects video bitrates via network conditions and the client's buffer occupancy. The traditional video streaming architecture is shown in Figure 1. The system consists of a video player client with a constrained buffer length and an HTTP-Server or Content Delivery Network (CDN). The video player client decodes and renders video frames from the playback buffer. Once the streaming service starts, the client fetches the video chunk from the HTTP Server or CDN orderly by an ABR algorithm. The ABR algorithm, implemented on the client-side, determines the next chunk and next chunk video quality via throughput estimation and current buffer utilization.After finishing the session, several metrics, such as total bitrate, total re-buffering time, and total bitrate change will be summarized as a QoE metric to evaluate the performance. Thus, how to achieve high QoE scores for adaptive video streaming has become a major challenge for ABR algorithms.

Existing ABR algorithms are generally composed of heuristics and learning-based. Heuristics make decisions from features with domain knowledge, e.g., throughput measured [13], buffer occupancy [7] or predefined models [8]. By contrast, learning-based ABRs model the process as the Markov decision process (MDP): at each step t, the video client, often namely *agent* in RL framework, take a proper action a_t (i.e.,



(a) User Heterogeneity

(b) Scenario Heterogeneity (3G, 4G, 5G)

Fig. 2. Visualizing personalized networks from the real-world [2], [20], [21]

TABLE I
COMPARISON RESULTS ON DIFFERENT ABRS OVER 3G-CAR AND 3G-BUS NETWORKS, WHERE A^{2}BR IS FINE-TUNED IN 20-SHOT.

Alg.	Bitrate (Mbps)(↑)	Time stalled (%)(↓)	Bitrate (Mbps)(↑)	Time stalled (%)(↓)
	3G-Car		*3G-Bus*	
BOLA	0.99 (3.9%↓)	1.47 (3.5%↑)	1.10 (97%↓)	1.18 (19%↑)
RMPC	1.09 (5.8%↑)	6.66 (3.7×↑)	1.87 (13%↓)	2.13 (1.2×↑)
Fugu	1.08 (4.8%↑)	5.54 (2.9×↑)	1.81 (16%↓)	2.11 (1.1×↑)
Pensieve	1.06 (2.9%↑)	5.47 (2.8×↑)	1.79 (17%↓)	2.12 (1.1×↑)
A^{2}BR	1.03	1.42	2.17	0.99

select a proper bitrate) w.r.t current system status s_t. The agent then downloads the chunk and computes a reward r_t for measuring the current quality-of-experience (QoE) of the past action. The process will terminate if the agent finishes playing the video session. In the end, we aim to generalize a policy π to maximize the QoE of the entire session.

The accumulated QoE objective function is defined as Eq. 1 ([8], [11]), where R_n represents the each chunk's video bitrate, T_n reflects the rebuffering time for each chunk n, $q(R_n)$ means the quality metric such as video bitrate [8] and VMAF [22] (state-of-the-art quality assessment), μ and σ are the weight of rebuffering and smoothness penalty, respectively.

$$\text{QoE} = \sum_{n=1}^{N} q(R_n) - \mu \sum_{n=1}^{N} T_n - \sigma \sum_{n=1}^{N-1} |q(R_{n+1}) - q(R_n)| \tag{1}$$

B. Different Types of Network Conditions

Recently, several learning-based schemes have been made to train *an NN policy* from the clean slate via various RL methods [11], [23]. Unfortunately, such one-fits-all schemes, including heuristics and learning-based can hardly always perform well in today's network traffics due to the diversity of real-world network conditions [2]. We show the personalized network environments from two perspectives.

Sorted by users: First, we measure a portion of data from the Puffer project [2] and demonstrate the users' personalized network status on June 2, 2021, in Figure 2(a). The left figure illustrates the correlations between throughput and round-trip-time (RTT) of each user. As shown, in the real world, the average bandwidth is particularly varied, ranging from 0.1 to 100 Mbps. The lower bandwidth leads to larger RTT. The network environment of each user is different. Someone can watch the videos with high bandwidth and low RTT, while the others live in the low bandwidth and high RTT scenario. The right figure plots the fine-grained cumulative distribution function (CDF) of throughput and RTT of the users with top-8 viewing hours on that day. We can find the tailor-made features for personalized network conditions: some of the users have very constant throughput (e.g., user C and user F), while most of the users' bandwidth is unstable and doesn't cover all network conditions.

Sorted by scenarios: Next, Figure 2(b) shows another personalized network condition that is categorized by network types, which covers 3G, 4G, and 5G networks. Testing results on the bus, car, and metro environments show that different vehicle speeds lead to very different 3G bandwidth distributions. For instance, we can see the throughput measured from the metro achieves the lowest average and fluctuation value among the candidates. While we observe the highest bandwidth with high fluctuation in the 3G-car scenario. Meanwhile, in addition to the various network specifics on 4G and 5G, the network distributions are always influenced by user behaviors: the network on walking and driving also have their particularity. Hence, the domain gap, which represents the relationship between network traffic distributions across different network types and users, has brought great challenges to recent rate adaptation algorithms.

ABR performance: How do existing one-fits-all ABR algorithms perform in such diverse but unique network conditions? Table I shows the average bitrate and stall ratio of existing ABR algorithms (§V-A) over different mobility types (car and bus) [20]. We show that the irregular networks greatly disturb the stability of the learning-based algorithm, since the difference between the network traffic distributions of the training set and the testing set. Moreover, heuristics like BOLA and RobustMPC (RMPC here) often perform well in one scenario but fail in the other, e.g., BOLA gains a low average bitrate and RobustMPC performs with a high stall ratio. Results indicate that the domain gap among heterogeneous network scenarios (e.g. Figure 2(b)) leads to the unstable performance of both heuristics and learning-based approaches [24], [16]. One of the feasible ways is to enable the policy to quickly adapt to the current network condition with few trials. As shown, our proposed method A^{2}BR outperforms existing techniques on video bitrate and stall ratio after being trained in 20-shot.

In summary, we argue that off-the-shelf "one-fits-all" ABR algorithms fail to provide acceptable performances for all users since the diversity of users' network conditions.

III. Methods

In this section, we start with modeling the tailored ABR process as an Input-driven Markov Decision Process (IMDP). Next, we explain why we have to construct a two-stage process rather than a vanilla one-stage approach. Finally, we briefly introduce meta-agnostic meta-learning and how to leverage domain knowledge.

A. Input-Driven MDP

Motivated by the observation above, we place the ABR problem in the discrete-time input-driven Markov decision process (MDP) [25], [26]. In detail, we consider the vanilla adaptive video streaming process: at each step t, the video client, often namely *agent* in RL framework, select a proper bitrate w.r.t current system status. The agent then downloads the chunk and computes an instant score for measuring the quality of the past action. The process continues until the agent finished playing the video session.

Definition 1. An input-driven MDP $\mathcal{M}$ is defined by a 4-tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{Z}, \mathcal{R})$, in which $\mathcal{S} \subseteq \mathbb{R}^n$ is a set of n-dimensional states observed (e.g., past throughput measured, buffer occupancy, past bitrate selected, etc.), $\mathcal{A} \subseteq \mathbb{R}^m$ is a set of m-dimensional actions, representing the bitrate candidates of next video chunks, $\mathcal{Z} = \{z_0, z_1, \dots\}, \subseteq \mathbb{R}^k$ is a set of k-dimensional input process, as $\mathcal{S} \times \mathcal{A} \to \mathbb{R}$ denotes the intermediate reward for each bitrate selection operation on the given state.

Commonly, the input process in the ABR problem is often denoted as a set of exogenous variables. For example, the personalized network traffic distribution for each user, network status in various network types, tailored QoE preference, etc. Notably, z_t is a general process, which is independent for the state s_t and action a_t. In other words, the a_t depends on s_t only, with no relationship to z_t – this is the key difference between input-driven MDPs and Partially Observable MDPs [27]. The reward function for ABR algorithms is often defined to achieve high quality of experience (QoE).

Definition 2. For an input-driven MDPs, the stochastic transition dynamics are given by

$$T_a(s'; s, z) = Pr(s_{t+1} = s'; s_t = s, a_t = a, z_t = z), \quad (2)$$

representing a state-transition probability of next state s_{t+1} with the given any state s_t, action a_t, and current personalized networks z_t.

Definition 3. Followed by the definition of input-driven MDP, the Q-value of a given state-action pair can be defined as

$$Q(s, a, z) = \sum_{s' \in \mathcal{S}} T_a(s'; s, z) \left(r(s, z, a) + \gamma V(s', z')\right) \quad (3)$$

Here $V(s', z')$ is the value function for state s', γ is the discounted factor $\in [0, 1)$.

When $\gamma < 1$, there exists an optimal policy $\pi^*(s, z)$:

$$\pi^*(s, z) = \arg\max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} T_a(s', z'; s, z) \left(r(s, z, a) + \gamma \max_{a' \in \mathcal{A}} Q(s', a', z')\right) \quad (4)$$

Here we consider two agents with the same policy π, while they work in the IMDPs with different input processes $\mathcal{Z}_1$ and $\mathcal{Z}_2$. When observing the same state s, the following agents would determine the same action a. Thus, the difference between the Q values of two agents will be equal only if $\mathcal{Z}_1$ equals $\mathcal{Z}_2$.

For solving Eq. 4, we can employ various reinforcement learning (RL) strategies if $\mathcal{Z}$ is known before the process starts. However, in practice, the agent cannot perceptualize its personalized network traffic before transmitting video streams. Assuming that the input process $\mathcal{Z}$ is "agnostic" for the agent, we find that vanilla RL method can only learn the optimal policy $\hat{\pi^*}$ which is relevant to $Q(s', a')$ instead of $Q(s', a', z')$:

$$\max_{a' \in \mathcal{A}} Q(s', a') = \mathbb{E}_{z' \sim T} \max_{a' \in \mathcal{A}} [Q(s', a', z')]. \quad (5)$$

There exists the variance reduction between the two cases, which eventually results in the sub-optimal policy [26]. Hence, we have a challenge here: *considering that the input process can hardly be explicitly observed, how to learn a tailor-made ABR algorithm for heterogeneous network conditions?*

B. Meta-RL with Domain Knowledge

With the rapid progress of on-device machine learning in both academia [28] and industry [29], training NNs on users' devices has already been a practical way of learning the tailor-made ABR policy from a clean slate *in situ*. Nevertheless, recent model-free RL technologies lack sample efficiency, which requires high convergence time on each client [2]. For example, a single agent requires at least 640,000 steps, spanning over 2 years, to converge in the real world. Most users would leave the platform before the algorithm has been completely trained [11].

In this paper, we consider a two-stage approach, which is composed of *offline stage* and *online stage*. Technically, at the offline stage, we attempt to train the *meta policy* via the traces collected by different network conditions, aiming at improving the average performance for all networks. At the online stage, we continually optimize the meta policy to fast "identify" the unique input process for *adapting to the personalized networks*. To achieve this, we encounter two new challenges based on the specific features of ABR tasks: *i) how to obtain a good parameter initialization for fast-learning? ii) How to efficiently learn tailor-made ABR algorithms online?*

Model agnostic meta-learning: for the first challenge, we present a method based on *meta-learning*, which provides an alternative paradigm to improve the learning algorithm itself and gains experience over multiple learning

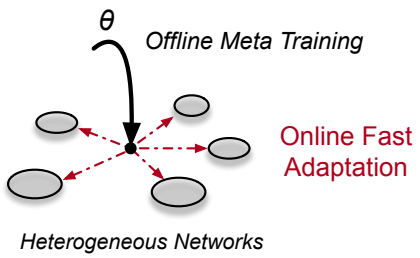


Fig. 3. The key principle of our method. We consider learning a good parameter initialization (θ), which can fast adapt to personalized networks.

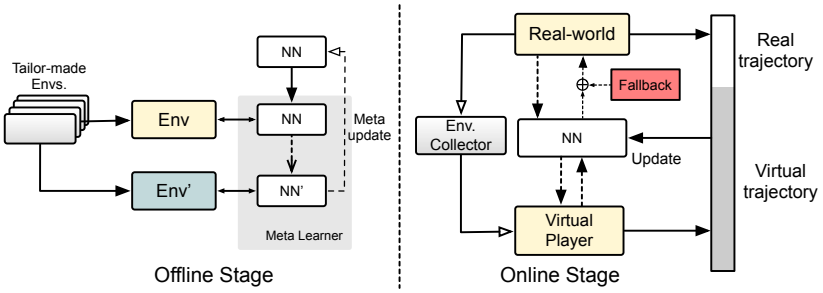


Fig. 4. The system overview of A^{2}BR . A^{2}BR mainly consists of two stages, the offline stage and the online stage.

episodes [30]. Treating the task as the user's personalized network environment, we find that model agnostic meta learning (MAML) [18] is quite suitable in personalized ABR scenarios where the network traces on each user are quite limited. More comparison of existing meta-learning methods is discussed in §VI-A. Specifically, MAML consists of an inner loop and an outer loop. For every cycle of the outer loop update, a specific task will be sampled from a distribution of tasks. and trains the parameter weights that determine the agent's behavior. In the inner loop, the agent interacts with the sampled environment and optimizes for maximizing the accumulated reward, i.e., QoE (Eq. 1).

Let θ denote the parameter weights, inner/outer loop learning rate are represented as α/β, and policy improvement function $\mathcal{L}$, for a distribution of task $\mathcal{T}$, the meta-optimization process can be presented as Eq. 6.

$$\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_\theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)) \qquad (6)$$

Leveraging domain knowledge: for tackling the second challenge, apart from the gains from MAML, we attempt to adopt the domain principle and knowledge of adaptive video streaming to accelerate the learning efficiency on the online stage. On the one hand, given a complete network trace, recent research has revealed that the ABR process can be precisely emulated by an ABR virtual player [10], [31]. Thus, based on the domain principles of the ABR framework, we implement a faithful ABR simulator to virtual rollout the trajectories, aiming to help improve data efficiency and generalization ability. On the other hand, we treat the domain knowledge of state-of-the-art heuristics [8] as the fallback policy which can help identify if the meta policy takes the system into the unexpected status (e.g., interrupt stall event). Putting them together, during the online stage, the agents continually optimize the meta policy according to the trajectories collected from both the real-world and the virtual player, while the real-world samples often account for a small part of them.

IV. A^{2}BR Overview

We propose A^{2}BR (Adaption of Adaptive BitRate), a novel neural ABR system that can quickly adapt the personalized network conditions via meta-RL and domain knowledge. The system workflow is shown in Figure 4. A^{2}BR consists of offline meta stage and online adaptation stage. At the offline stage, we train a meta-model using MAML with various network environments to learn a good *parameter initialization* for achieving both acceptable "mean" performance and fast adaptation. At the online stage, the agents continually tune the meta-model with the help of domain knowledge for rapidly varying the personalized network condition, i.e., generating a tailor-made ABR algorithm.

A. Basic Training Algorithm

In this section, we introduce the NN architecture for each model in A^{2}BR . First, we describe the NN's inputs, outputs, and architecture. Then, we explain the basic training methodology of A^{2}BR .

1) NN Model Overview: The NN architecture is shown in Figure 5. Here we denote the parameters of the meta actor model as θ_π and the meta critic model as θ_v. What's more, we refer to the combination of the meta actor model and critic model as the meta actor-critic model.

Inputs. As mentioned before, A^{2}BR is allowed to continually learn the system dynamics at the online stage, which motivates us to consider the computational overhead during the inference phase. In other words, A^{2}BR 's input should be carefully designed by avoiding trivial features. In the beginning, we train a teacher network with all possible features as the input (e.g., past bitrate, buffer throughput, download time, response time, bitrate map, chunk map, chunk remaining). Next, we use light weighted machine learning model, i.e., decision tree, to imitate the NN's policy and prune the most trivial features [32]. Finally, our state representation is listed as follows.

For each video chunk t, the agent takes 5 metrics, totally 17 critic features, as the state s_t. The state contains past video quality q_t, current buffer occupancy b_t, past k chunk's throughput measured, i.e., C_t, past k chunk's download time, i.e., D_t, and past k chunk's response time: i.e., P_t. Hence, the state s_t can be written as $\{q_t, b_t, C_t, D_t, P_t\}$. We set past k as 5 for further reducing the state size due to the light-weighted requirements. Moreover, instead of feeding the exact values of gathered statistics to the agent, we also use normalized statistics. The state normalization method enables the agent to generalize the strategy better in unseen network environments [33].

Outputs. The A^{2}BR 's actor model uses a discrete action space, i.e., an n-dim vector, which indicates the probability

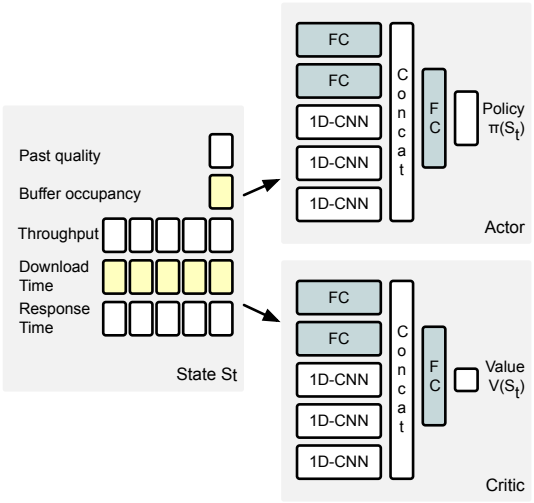


Fig. 5. A^{2}BR 's NN architecture overview. A^{2}BR consists of an actor network and a critic network.

of the bitrate level being selected under the current state. The A^{2}BR 's critic model outputs a single scalar, representing the estimated value for the current state.

NN architecture A^{2}BR uses a neural network (NN) to take an action for the given state. For each video chunk, the agent mainly takes the past five values as a sequence for representing the current state, including past video quality, buffer occupancy, throughput, download time, and response time. As shown in Figure 5, the A^{2}BR 's NN architecture uses several Conv-1D layers and fully-connected layers to extract features. In detail, we first use three Conv1D layers with feature number=64, and kernel size=1 to extract features from throughput, download time, and response time. Meanwhile, we adopt two fully-connected layers with feature number=64 to up-sample the features of past video quality and buffer occupancy. Then we use a concatenate layer to concentrate all the features and take a fully connected layer with 64 neurons to down-sample the features. Finally, we take an n-dim vector with Softmax activation function to represent the actor network's output and use a single scalar to represent the critic network's output.

2) Maximum Entropy PPO: As mentioned before, the basic idea of DRL is to improve the policy via improving the probabilities of the high-reward-samples and avoiding the possibilities of the failure-samples from the sampled trajectories. In other words, the improved policy π at state s_t is required to pick the action a_t which produced the best-accumulated reward R_t, i.e., $a_t = \text{argmax}_a \, \mathbb{E}_t[R_t(s_t, z_t, a)]$.

Due to the setting of meta-learning (§III-B), A^{2}BR often requires more exploration at the offline stage, while less exploration but more exploitation at the online stage. To that end, inspired by the recent maximum entropy policies [34], we present ME-PPO (Maximum Entropy Proxy Policy Optimization) to train the NN. See in Eq. 7, the improved policy π_θ at state s_t is required to pick the optimal action ${a_t}^*$ which produced the best accumulated reward R_t=$\sum_t \gamma^t(r_t + \lambda H^{\pi_\theta}(s_t))$, in which $H^{\pi_\theta}(s_t)$ is the entropy of the current policy (Eq. 8), λ is the entropy weight which encourage exploration feedback. It is strongly correlated to the unpredictability of the actions which an agent takes in a given policy. The greater the entropy, the more random the actions that an agent performs, and vice versa.

$$ {a_t}^* = \arg\max_a \hat{\mathbb{E}}_t[\sum_t \gamma^t(r_t + \lambda H^{\pi_\theta}(s_t))] \tag{7} $$

$$ H^{\pi_\theta}(s_t) = -\sum_{i \in A} \pi_\theta(a_i; s_t) \log \pi_\theta(a_i; s_t). \tag{8} $$

ME-PPO is incrementally implemented based on state-of-the-art on-policy DRL algorithm Dual-clip Proxy Policy Optimization (Dual-PPO) [35]. Briefly, the Dual-clip PPO algorithm adopts a double-clip method to restrict the step size of the policy iteration and update the NN by minimizing the following *clipped surrogate objective*.

The loss function of the A^{2}BR 's actor network is computed as Eq. 10,

$$ \begin{aligned} \mathcal{L}^{PPO} = \min \Big(& \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}(\theta)\hat{A}_t, \\ & \text{clip}\big(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}(\theta), 1-\epsilon, 1+\epsilon\big)\hat{A}_t \Big). \end{aligned} \tag{9} $$

$$ \mathcal{L}^{\textit{ME-Policy}} = \begin{cases} \hat{\mathbb{E}}_t[\max(\mathcal{L}^{PPO}, c\hat{A}_t)] & \hat{A}_t < 0 \\ \hat{\mathbb{E}}_t[\mathcal{L}^{PPO}] & \hat{A}_t \geq 0 \end{cases} \tag{10} $$

where $\hat{A}_t$ is the advantage function:

$$ \hat{A}_t = r_t + \gamma[V^{\pi_\theta}(s_{t+1}) + \lambda H^{\pi_\theta}(s_{t+1})] - V^{\pi_\theta}(s_t). \tag{11} $$

Here ϵ and c are hyper-parameters that control how to clip the gradient. We set $\epsilon = 0.2$, $c = 3$ as consistent with the original paper [35].

The A^{2}BR 's critic network V_{θ_p} is updated via minimizing the error of the advantage function $\hat{A}_t$: $\mathcal{L}^{\textit{Value}} = \frac{1}{2}\hat{\mathbb{E}}_t\left[A_t\right]^2$. We summarize the loss function $\mathcal{L}^{ME-PPO}$ in Eq. 12.

$$ \nabla\mathcal{L}^{ME-PPO} = -\nabla_\theta \mathcal{L}^{\textit{ME-Policy}}(\pi_\theta, \hat{A}_t) + \nabla_{\theta_v}\mathcal{L}^{\textit{Value}}. \tag{12} $$

Meanwhile, considering that on-policy RL is sensitive to the entropy weight and it usually requires careful tuning [10], we autonomously adjust the entropy weight λ for minimizing the gap between the current entropy and the target entropy H_{target} (Eq.13). We set $H_{target} = 0.1$ as suggested by related work [36]. α is the learning rate of the actor-network.

$$ \lambda \leftarrow \lambda - \alpha\left[H^{\pi_\theta}(s_t) - H_{target}\right]. \tag{13} $$

We summarize all hyper-parameters of ME-PPO as follows: i) entropy weight λ, ii)PPO clip factor ϵ, iii) Dual-clip PPO clip factor c, iv) target entropy H_{target}, and v) learning rate α, β. It's important that most parameters (i.e. $\epsilon, c, \alpha, \beta$) are configured as the default settings of the original paper [17], [35], [36]. The only special parameter is the entropy weight λ. It is dynamically being tuned by H_{target} and α during training.

Algorithm 1 Meta-learning for the Offline Stage

Require: $p(\mathbf{Env})$: distribution over heterogeneous networks.
Require: α, β: learning rate for inner-loop and outer-loop.
randomly initialize ψ, θ_π, and θ_v
while not done **do**
  Sample user's network environments $\mathbf{Env}_i \sim p(\mathbf{Env})$
  // Train the NN in parallel, agent number K
  for $\mathbf{Env}_i$ in K **do**
    // Inner Loop Phase
    Rollout M trajectories $\mathcal{D}$ in $\mathbf{Env}_i$ using f_θ.
    Meta-update using using ME-PPO: $\theta_i' \leftarrow \theta - \alpha \nabla_\theta L^{ME-PPO}_{\mathbf{Env}_i}(f_\theta, \mathcal{D})$.
    // Outer Loop Phase
    Sample trajectory $\mathcal{D}_i'$ using $f_{\theta_i'}$ in $\mathbf{Env}_i$.
  end for
  // Outer Loop Update
  Update θ with $\hat{D} = \{\mathcal{D}_i' | i = 1, \cdots, K\}$ using policy gradient: $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathbf{Env}_i \sim p(\mathbf{Env})} L^{PG}_{\mathbf{Env}_i}(f_{\theta_i'}, \mathcal{D}_i')$.
end while

B. Meta-Learned Policies for Offline Stage

Inspired by vanilla MAML methods, the offline training phase can be categorized into the inner loop phase and outer loop phase (Alg. 1). In the inner loop phase, for each epoch, the worker i first randomly picks a specific network condition as the environment from the network status pool, and samples N trajectories in that environment according to the current policy π_θ. Then the meta-model is optimized by the collected trajectories with the ME-PPO method. Here we treat the learned meta-model as θ_i'. In the outer loop phase, the worker i continually rollouts several trajectories from the randomized selected environments with the meta-policy $f(\theta_i')$, and computes gradients for θ with the trajectory. Subsequently, each worker sends the computed gradients to the central agent. The central agent finally merges the gradients via workers' loss functions and the outer loop's learning rate β. In addition, we make the training phase of the online stage more practical from different perspectives.

Meta-learned value network. First, we adopt *fresh* trajectories to adapt the meta value network before updating the meta policy network. Such settings allow the framework to estimate the advantage function precisely and avoid introducing extra bias caused by exogenous inputs to the baseline [26].

Policy gradient for the outer loop. Next, we focus on policy gradient methods [37] for expressing the loss function of the outer loop to accelerate the training process, since there's no obvious distinction in the overall performance between the complex ME-PPO loss and the vanilla policy gradient loss in the outer loop. In turn, we keep using ME-PPO in the inner loop due to its advantages compared with the policy gradient method (Eq. 14).

$$\mathcal{L}^{PG} = -\mathbb{E}_t \left[\nabla_\theta \log \pi_\theta(at; st) \hat{A}_t \right] \tag{14}$$

Training with First-Order MAML. Finally, we simplify the MAML process to the First-Order MAML, which computes the meta-objective derivative at the post-update parameters directly [38]. In brief, first-Order MAML ignores the second derivative part and doesn't have to use all the inner gradients for updating.

Algorithm 2 Learning Tailor-made ABRs for the Online Stage

Require: θ: The trained meta-model in the offline stage.
Require: $\mathcal{D}_{\mathbf{Env}}$:the collection of network environments experienced.
$\mathcal{D}_{\mathbf{Env}} = \{\}$.
for video session **do**
  // Rollout policy with the "real" player.
  t $\leftarrow$ 0; $\mathcal{D} = \{\}$.
  while not done **do**
    Get ABR state s_t.
    Get π w.r.t s_t and θ: $\pi_\theta(s_t)$.
    Predict future throughput: $\hat{c}_t$.
    Generate mask m with the fallback policy (Eq 16).
    Pick $\hat{a}_t$ according to Eq. 15.
    Calculate instant reward r_t.
    Add $\{s_t, \hat{a}_t, r_t\}$ to $\mathcal{D}$.
    t $\leftarrow$ t+1
  end while
  Estimate environment experienced Env from $\mathcal{D}$ in hindsight; Add Env to $\mathcal{D}_{\mathbf{Env}}$.
  // Rollout policy from virtual environment.
  for M Rollouts **do**
    Uniformly sample condition Env' from $\mathcal{D}_{\mathbf{Env}}$.
    Rollout trajectories $\mathcal{D}'$ using Env' and $f(\theta)$.
    Add $\mathcal{D}'$ to $\mathcal{D}$.
  end for
  Update meta-model θ with $\mathcal{D}$.
end for

C. Learning Tailor-made ABRs for Online Stage

Recall that we attempt to learn a tailor-made ABR algorithm for varying current heterogeneous networks within few-shot learning at the online stage. As much as MAML enables the meta models (i.e., NN) to learn quickly for varying the current users' network condition, it still takes at least 2,000 times on watching videos to complete the adaptation to the current network (even it's 320× faster than the prior approach that learns from clean slates). Hence, we leverage domain principles and knowledge, such as virtual environment replay and safe exploration for online RL, to further improve the learning efficiency in the online stage.

More specifically, the online training process is mainly composed of a learner, an environment collector, and a fallback policy. The pseudocode in Alg. 2 depicts the overall algorithm. When the video session starts, the video player receives the trained meta model θ_π from the training server. Then the player makes the ABR decision with the combination of the meta policy and the fallback policy. Such a hybrid decision enables the player always to play on the "safety" track.

Fallback policy design: The pre-trained meta-model learns parameter-initialization for varying different network con-

ditions, while it hardly guarantees the robustness of our system at the online stage. The meta-model is required to be continually trained at the online stage, as unsafe bitrate decisions may still happen due to action explorations or unexpected changes in the network environment. Hence, we have to design a proper fallback policy to avoid unnecessary stalling events caused by exploration. However, revisiting the recent safe and robust online RL approach [39], [40], [41], [42], [43], we find that none of the schemes can satisfy our requirements. For example, reward shaping-based approaches such as OnRL [41] and Deep-OR [40] integrate an instinct reward signal as a switching penalty into the reward function. While in our case, the reward functions of the offline stage and the online stage must be consistent, otherwise, the critic network has to be retrained w.r.t the changed reward function. Such inaccurate value estimations will eventually break the fast learning. Meanwhile, other vanilla mask-based approaches (e.g., Decima [42]) block the unsafely or invalid actions by applying action masking [44]. Nevertheless, the mask values are often determined by the exogenous inputs, e.g., global DAG information ([42]), which is not included in the state space. Hence, the tailor-made ABR policy cannot be successfully learned without changing the state representation.

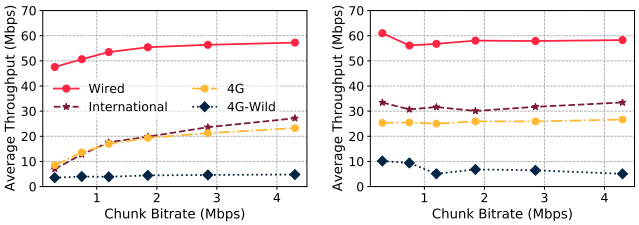
To this end, we propose a fallback policy that only relies on the metrics in the current state representation, that is, it doesn't require any additional modification to the reward function. The fallback policy is a hybrid scheme that combines the original NN's actor outputs and the mask of a heuristic-based method, shown in Eq. 15. The key principle of the mask is to "filter" out all the bitrate actions that *might incur rebuffering events*, instead of directly making bitrate selection policies. As listed in Eq. 16, the heuristic-based method is motivated by HYB [10] that simply picks the maximum bitrate without occurring stall events. In detail, at chunk t, the hybrid action $\hat{a}_t$ for the given state s_t becomes

$$\hat{a}_t \sim \{\frac{m_i e^{w_i}}{\sum_j m_j e^{w_j}} | \forall i = [1, \ldots, j]\} \tag{15}$$

$$m_i = \begin{cases} 1 & \hat{c}_t b_t - R_i L > 0 \\ 0 & \text{otherwise}, \end{cases} \tag{16}$$

where for the i-th mask of the total bitrate levels (aka., actions) j, m_i indicates a mask, presented by a $\{0,1\}$ vector, that controls whether the action a_i are safe or not. $\hat{c}_t$ is the predicted network capacity. Here $\hat{c}_t$ is calculated by the average value of past throughput measured, i.e., $\hat{c}_t = \sum C_t / \sum (D_t + P_t)$. Note that it can be measured by any prediction method, such as EWMA and harmonic mean. b_t is the current buffer occupancy of the player. Each chunk has the same video time of L seconds. Thus, $R_i L$ means the average chunk size for the $i-th$ bitrate-level. w represents the final NN output with no activation functions. The mask values can be easily computed from the state input. As a result, we can still use the original training techniques to update the NN since the back-propagation of the gradient of the NN still holds.

We evaluate the proportion of using the original meta-policy and fallback policy. Results show that the fallback policy only accounts for about 1%-4% of the overall decision-making in different network scenarios (not shown). It makes sense since the fallback policy will be enabled *only if the meta-policy picks the bitrate that might occur during the stall event*. In other words, the fallback policy is the lower bound of A^{2}BR . Most of the decisions are still determined by the meta-policy model.



(a) Simulation without RTT. (b) Simulation with RTT.

Fig. 6. Our virtual player focus on the RTT dynamics (right side), which leads to better simulation results.

Consequently, the player safely rollouts the trajectory $\mathcal{D}$ w.r.t the hybrid policy. Upon finishing the session, we "restore" the current network Env from $\mathcal{D}$ and put it into the environment collector $\mathcal{D}_{\mathbf{Env}}$. Then we randomly sample M network environments from the collector and use the virtual player to roll out another set of trajectories $\mathcal{D}'$. Finally, the learner employs ME-PPO for meta policy training according to $\mathcal{D}$ and $\mathcal{D}'$. We discuss the best learning epoch for A^{2}BR in §VI-B.

V. Evaluation

In this section, we evaluate A^{2}BR in several personalized network environments, including user-personalized, 3G, 4G, and 5G networks, where the average bandwidth of which are gradually increased, ranging from 3 Mbps to 110 Mbps. Furthermore, we enhance A^{2}BR to support varying the QoE of user preferences. Finally, we conduct a real-world experiment to understand the generalization of A^{2}BR .

A. Methodology

Implementation. The A^{2}BR 's gym-like environment and the virtual player are written by Python 3.6. At the same time, we adopt TFLearn 1.5.0 [45] to build the A^{2}BR 's NN and TensorFlow 2.4.0 [46] to implement the training workflow. We set inner loop's learning rate $\alpha = 10^{-4}$, outer loop's learning rate $\beta = 10^{-3}$, virtual player rollout M=20. Meanwhile, we use Adam [47] to optimize the model.

Testbed. We build a trace-driven "gym"-like [48] simulator to train and validate A^{2}BR w.r.t various network datasets and video sets. The simulator is pragmatically implemented based on various state-of-the-art ABR virtual simulators [49], [14]. Moreover, we integrate round trip time (RTT) into the simulator for improving the accurateness of throughput measurement.

Simulator fidelity. Now we show the strength of our proposed virtual player. Previous studies demonstrate that the used congestion control algorithm can impact the performance

TABLE II
RESULTS OF EXPERIMENT IN DIFFERENT VEHICLES, SUMMARIZED IN VMAF [22] AND STALL TIME.

Alg.	Car		Bus		Ferry		Metro	
	VMAF(↑)	Time stalled (%)(↓)	VMAF(↑)	Time stalled (%)(↓)	VMAF(↑)	Time stalled (%)(↓)	VMAF(↑)	Time stalled (%)(↓)
BOLA	67.42	1.47	71.21	1.18	66.09	5.68	64.78	3.88
RobustMPC	63.18	6.66	74.43	2.13	59.90	2.65	60.71	2.06
Oboe	67.80	4.10	76.33	1.94	64.08	2.71	61.58	1.98
Fugu	66.41	5.54	76.59	2.11	59.96	1.96	61.33	1.82
Pensieve	65.83	5.47	72.34	2.12	60.23	3.14	59.14	1.79
Comyco	67.68	3.74	77.17	1.84	63.24	2.28	62.59	2.39
A^{2}BR (VMAF)	68.65	1.42	83.78	1.03	60.71	1.45	63.34	1.79

of ABR algorithms due to the cross-layer effects, as one of the better solutions is to faithfully measure the round-trip propagation time at each time [50]. At the same time, round trip time (RTT) is also observable, since the video client can estimate the current RTT via estimating time-to-first-byte (TTFB) or *response time* [4]. To that end, we apply the RTT dynamic module to the simulator for enhancing the throughput prediction. We conduct 4 real-world experiments to prove the effectiveness of our simulator. Specifically, we propose a Round-Robin ABR algorithm that picks the bitrate orderly and uses the algorithm to collect the information for each chunk, such as download chunk size, download time, and current RTT. After finishing the video session, we compute the throughout for each bitrate using RTT or not using RTT respectively. Note that we finish the experiment in *stationary* network environments, in which their network capacity does not change rapidly. Results are illustrated in Figure 6. We can see that estimating throughput with RTT performs much better than that not using the RTT metric.

ABR Baselines. We select several representational ABR algorithms, which include heuristics and learning-based. All the baselines are retrained or tuned for fitting each experiment.

- **BOLA [51]:** a popular buffer-based heuristic that turns the ABR problem into a utility maximization problem and solves it by using the Lyapunov function.
- **RobustMPC [8]:** the state-of-the-art heuristics which consider both the buffer occupancy and throughput predictions and maximize the QoE by solving an optimization problem. BOLA and RobustMPC are still top-2 methods that are widely deployed in industries [4], [31].
- **Fugu [2]:** an ABR algorithm that leverages deep neural network (DNN) to estimate download time for each chunk, and uses model predictive control (MPC) to make decisions according to the estimated values. We retrain the DTP model of Fugu with partial information since it requires some TCP metrics which is not fully logged in the dataset.
- **Pensieve [11]:** an RL-based algorithm which takes the former network status as states and optimizes itself with various network conditions using A3C method [52]. In this work, we retrained Pensieve with our videos, datasets, and QoE metrics.

B. Case Study: Different Vehicles

First, we evaluate A^{2}BR and existing ABR algorithms in the personalized network scenarios which consider the various type of vehicles, including the car, bus, ferry, and metro.

1) Network and Video Settings: For the training set, we adopt a Markovian model where each state represented an average throughput in the range of 0.1-6Mbps [11]. For each epoch, we randomly sample trajectories with different initial parameters and take them as the virtual personalized scenario. In the online stage, we use HSDPA [20], a well-known 3G/HSDPA to continually train the meta-model. The network scenarios are naturally categorized into bus, car, ferry, and metro. Meanwhile, we use *EnvivioDash3* to evaluate existing algorithms, where the video chunks are encoded as $\{0.3, 0.75, 1.2, 1.8, 2.8, 4.3\}$ Mbps [4]. Besides that, we optimize A^{2}BR with the quality-aware QoE metric $\mathtt{QoE}_v$, which is constructed by video quality, rebuffering time, positive and negative smoothness [14] (Eq. 17). In this experiment, we set the maximum buffer size=60 seconds.

$$\begin{aligned}\mathtt{QoE}_v = \alpha_v \sum_{n=1}^{N} q(R_n) - \beta_v \sum_{n=1}^{N} T_n + \gamma_v \sum_{n=1}^{N-1} [q(R_{n+1}) - q(R_n)]_+ \\ - \delta_v \sum_{n=1}^{N-1} [q(R_{n+1}) - q(R_n)]_-\end{aligned} \tag{17}$$

Here α_v, β_v, γ_v, δ_v are the parameters to describe their aggressiveness. Followed by the original paper [14], we set $\alpha_v = 0.8469$, $\beta_v = 28.7959$, $\gamma_v = 0.2979$, $\delta_v = 1.0610$. Furthermore, for the sake of fairness, we also comparing the A^{2}BR with additional ABR approaches:

- **Oboe** [10], an auto-tuning mechanism that detects changes in network conditions and adjusts ABR's hyper-parameters according to the configured map. Since the official implementation of Oboe is not publicly available, we have tried our best to reproduce Oboe. For more detail please refer to [53].
- **Comyco** [14], a quality-aware ABR scheme that leverages imitation learning to improve the policy. We adopt the pre-trained model provided by the authors.

2) A^{2}BR vs. Existing Algorithms: We list the comparison results in Table II, where the video quality is measured as

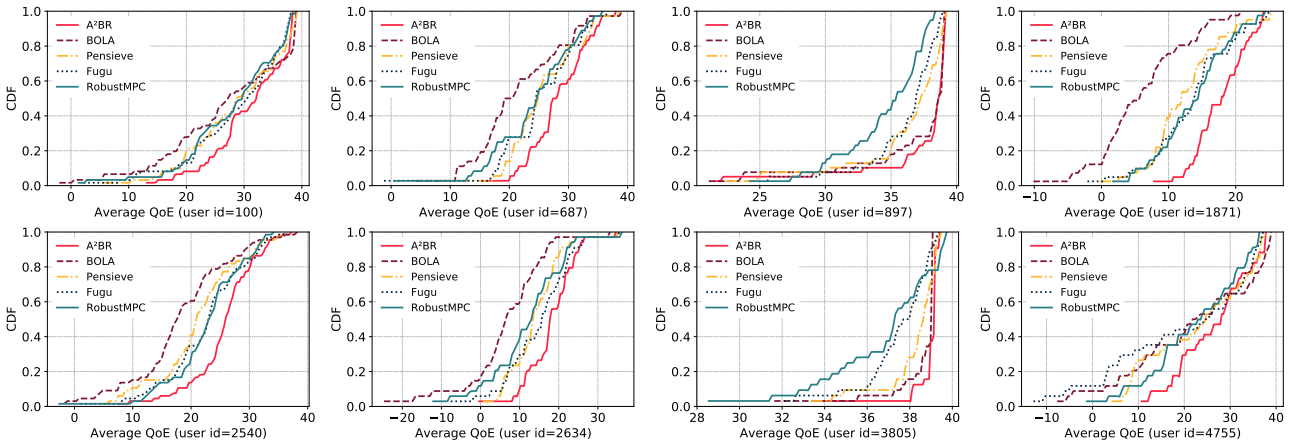


Fig. 7. Comparison results of user-personalized network conditions. In detail, we select top-8 users among 1000 unique users from the Puffer network dataset. Results are plotted with the CDF of QoE metrics.

VMAF [22]. Here we can see that A^{2}BR gains the highest VMAF score while guaranteeing the lowest stall time in Car and Bus scenarios. In particular, A^{2}BR improves the average VMAF by up to 12.6% and reduces the average stall time by 78.8% compared with RobustMPC. A^{2}BR also performs better than Pensieve with the average VMAF of up to 12.6% and the stall time of up to 69.3%. Furthermore, we observe that although A^{2}BR doesn't achieve the best performance on both VMAF metric and stall time in the Ferry and Metro network scenario. Here we make a deeper analysis as follows:

① Our first observation is A^{2}BR reaches the lowest stall time among all candidates and its VMAF comes only last followed by BOLA. However, BOLA performs 1.1×-2.8× higher stall time than that of A^{2}BR , which is surely a considerable price. In particular, BOLA often occurs rebuffering events (i.e. 5.68% in terms of the stall ratio) under the Ferry scenario. With further analysis, we observe that the network throughput of the Ferry scenario is highly variable. Traditional model-based approaches RobustMPC estimates the throughput cautiously, requesting chunks at median-level bitrates with keeping a medium-sized playback buffer. While BOLA mainly takes current buffer size as inputs, failing to perceive heavy ramp-down or ramp-up on the throughput. Hence, BOLA strongly prefers HD videos but occurs a high stall ratio in the Ferry scenario. In contrast, A^{2}BR attempts to reduce the rebuffering time rather than picking chunks with higher bitrates. Comparing the performance of A^{2}BR and Pensieve, we can see that A^{2}BR maintains the behaviors on average quality but further decreases 1.69% on relative time stalled.

② Second, comparing A^{2}BR with Fugu in 4 scenarios, we observe that, although Fugu leverages accurate prediction to exceed RobustMPC, Fugu also underperforms A^{2}BR in low-speed vehicles, such as the car and bus. In such scenarios, accurate throughput prediction hardly influences QoE. While Fugu almost matches the performance of A^{2}BR on both VMAF and stall ratio, which indicates the strength of throughput prediction.

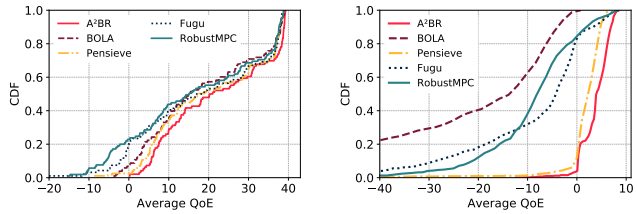
③ Finally, Oboe and Comyco show similar behavior over all considered scenarios, i.e., gaining higher video quality but slightly increasing the risk of rebuffering. The average time stalled for A^{2}BR is 62% lower than Comyco and 65% lower than RobustMPC on the car scenario. One of the reasons is that these schemes heavily depend on shortening the gap between the network distributions of the training set and the test environment. Same conclusions have also been observed in the results of Pensieve. A^{2}BR employs the online learning phase to "understand" and vary current network conditions, which eventually yield better performance.

C. Case Study: User-Personalized Networks

Next, we compare the performance of A^{2}BR with baselines in the user-personalized network conditions.

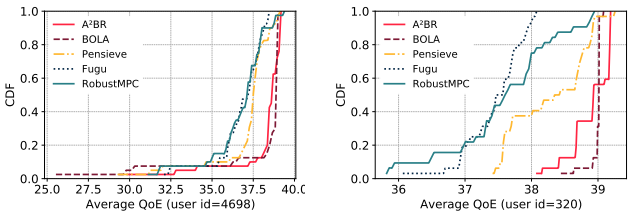
1) Network and Video Settings: We use the Puffer network dataset [2], which includes 580,708 real-world *wired* network environments collected from 28089 unique users. We randomly sample 1000 unique users from the dataset as the user-personalized network dataset. Each user contains at least 30 unique network traces and each trace lasts over 300 seconds. We split the dataset into two groups, where 80% of the dataset for training and 20% of the dataset for testing. Considering the wide range of users' network bandwidth, we use a 4K DASH dataset provided by Quinlan et. al. [54] as the video description. The videos are encoded at 13 bitrate levels, ranging from 0.235Mbps to 40Mbps (i.e., 40Mbps, 25Mbps, 15Mbps, 4.3Mbps, 3.85Mbps, 3Mbps, 2.35Mbps, 1.75Mbps, 1.05Mbps, 750Kbps, 560Kbps, 375Kbps, 235Kbps). We set $\mu = 40, \sigma = 1$ to balance the conflicting goals in QoE (Eq. 1), which is consistent with the maximum bitrate of the video. In this part, we set the maximum buffer size as 40 seconds.

2) A^{2}BR for different users: To better understand the A^{2}BR 's performance on each user, we report the detailed QoE breakdown of ABR algorithms over top-9 user-personalized network conditions in Figure 7, in which the user id represents the user logged in the Puffer dataset. As shown, BOLA performs well in fast-network scenarios with a small bandwidth jitter, such as user No. 897 and 3805. Especially in user No. 3805, BOLA even achieves the same performance compared with A^{2}BR . While it fails to work



(a) Primary experiment (b) Slow-network users

Fig. 8. Comparing the performance of A^{2}BR with existing ABR approaches in the user-personalized network conditions. **(Primary experiment: 650 stream-hours, Slow-network users: 117 stream-hours.)**



(a) Case A (b) Case B

Fig. 9. Demonstrating bad cases of A^{2}BR . A^{2}BR perform even worse than before if continually trained over such network conditions.

well in slow-network scenarios, e.g., user No. 1871. One of the underlying reasons is that existing heuristics, including BOLA and RobustMPC, require proper parameter settings to vary different network conditions. Although Oboe [10] considers using an auto-tuning method to help prior ABR algorithms for fitting different throughput regimes, it lacks fast adaptation ability. Specifically, Oboe adopts the offline mapping method to generate tailor-made ABR strategies for each network state. However, considering that computing the best parameter configuration for one network state takes about 12 seconds on a single core, computing a user-personalized network condition will take approximately 3 hours to explore with 1 core [10]. Thus, it's quite impractical to online map the best parameter for each user. The same conclusions are also observed in lifelong learning-based ABRs such as Comyco [24] and Fugu [2]. Comyco requires an hour to retrain the global policy as Fugu lasts a day to refresh the model. Especially, both Comyco and Fugu are interested in providing good video delivery QoE for all users instead of on average. In Figure 7 we can see that Pensieve does perform well across all considered network scenarios while it seldom performs the best performance among all baselines. In contrast, A^{2}BR rivals or outperforms other ABR schemes for most users in 20-shot.

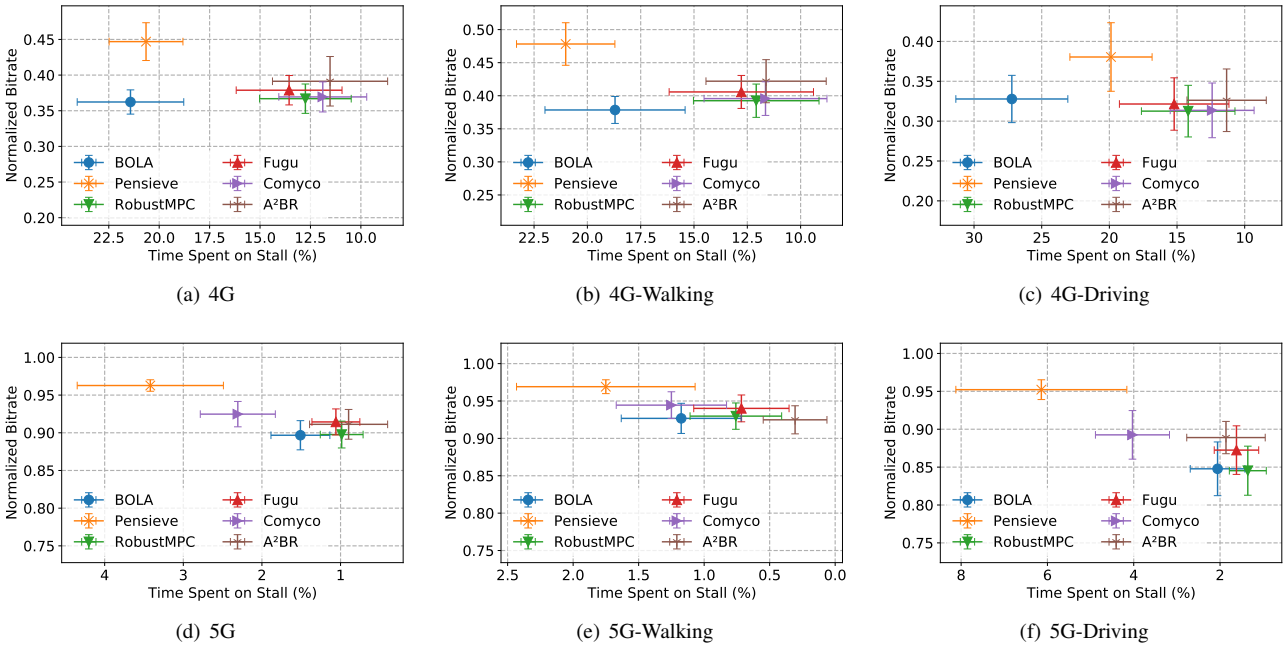
3) A^{2}BR vs. Baselines: Figure 8(a) shows the QoE breakdown of A^{2}BR and recent baselines. The results are evaluated on 650 stream-hours of network data. The performance gain in QoE between A^{2}BR and existing heuristic baselines is approximately 51% (BOLA) and 21% (RobustMPC). As expected, we also find that A^{2}BR also outperforms recent learning-based approaches, with the improvements on QoE of 12% on Pensieve and 21% compared with Fugu. It makes sense since existing ABRs didn't consider the input process, while such schemes will eventually fail if the current personalized network behaves differently from the fixed training network set. What's more, we report the breakdown results performing on the slow network users, where the users have an average throughput of less than 10Mbps. Such typical low-bandwidth scenarios are quite challenging for ABR algorithms [5]. As shown in Figure 8(b), we can see a significant benefit from using meta-RL for fast adaptation. Especially, the gap between A^{2}BR and Pensieve widens to 51% for average QoE. One of the reasons is A^{2}BR pays more attention to avoiding stall time (0.26% vs. 0.49%), which is 1× lower than that of Pensieve. Another possible reason is that, for most network conditions in the Puffer dataset, the ABR algorithm can blindly pick the chunk with the highest bitrate if the current measured throughput is significantly sufficient for downloading all bitrate levels [2]. Hence, recent RL technologies lack sufficient generalization abilities to handle such a "large" action space, i.e., 13 bitrate levels, which is 2× larger than the original version [11] A^{2}BR can solve this issue via learning environments *in situ*.

4) Deep dive: Upon analyzing the advantage of A^{2}BR , we investigate the lower bound of the proposed scheme, i.e., in which scenario that A^{2}BR performs worse than we expected? Figure 9 shows two bad cases of A^{2}BR . Case A (Figure 9(a)) indicates that A^{2}BR doesn't reach the best QoE among all ABR algorithms. Similar to user No. 897 in Figure 7, the ABR algorithm only needs to take the current buffer as the input to control such networks, since its network throughput is highly variable, which has negative impacts on QoE. A^{2}BR cannot learn such a complex logic in rather few-shot. Moreover, Case B (Figure 9(b)) shows a good network condition, as most ABRs achieve maximum QoE. A^{2}BR keeps exploring environments during the online phase, which may lead to the fair but not the best performance.

D. Case Study: 4G and 5G

In this part, we set up an experiment to understand how A^{2}BR performs in different wireless networks such as 4G and 5G, and how about walking or driving in those networks.

1) Network and Video Settings: We take the Lumos5G dataset [21], containing 121 5G and 175 4G throughput traces, collected at 1-second granularity. As suggested by [56], we formally categorize the network conditions into 4 types, i.e., 4G with walking, 4G with driving, 5G with walking as well as 5G with driving. Considering the wide range of 5G's bandwidth, we use a 4K video encoded as {20, 40, 60, 80, 110, 160}Mbps [57]. Motivated by the prior study [56], we also modify the QoE metric $\mu = 160, \sigma = 1$, where μ is often set as the maximum number of bitrate levels of the 4K video. Here we configure the maximum buffer occupancy as 60 seconds. Same as §V-B, we take Comyco [14] as the baseline. Note that Comyco is retrained with the aforementioned QoE objective. Besides, considering that the general linear-based reward function (Eq. 1) might hardly map the actual QoE for 4K videos [23], [56], we leverage a more realistic QoE model, i.e., ITU-T Rec. P.1203 [55] for evaluating the performance.



(a) 4G (b) 4G-Walking (c) 4G-Driving

(d) 5G (e) 5G-Walking (f) 5G-Driving

Fig. 10. Comparison of average normalized bitrate and stall ratio in 4G and 5G. Error bars show 95% confidence intervals.

TABLE III
QOE PERFORMANCE COMPARISON OF DIFFERENT ABR ALGORITHMS. RESULTS ARE CALCULATED BY ITU-P.1203 [55].

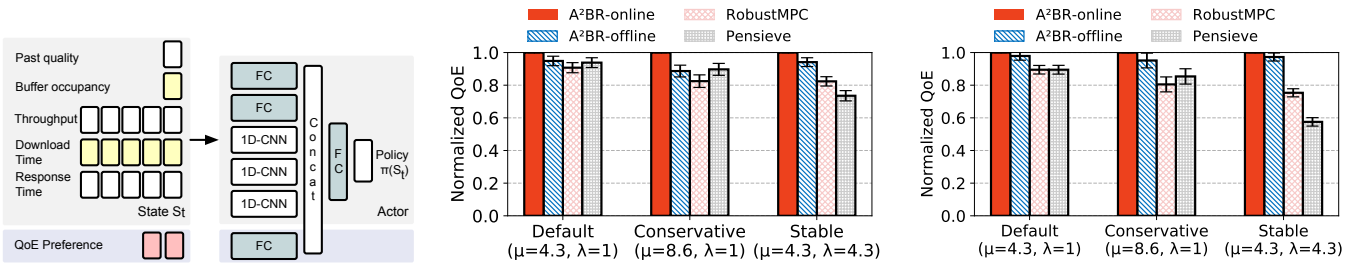
Alg.	4G			5G			All		
	O.23(↑) (Stall)	O.35(↑) (Visual)	O.46(↑) (Overall)	O.23(↑) (Stall)	O.35(↑) (Visual)	O.46(↑) (Overall)	O.23(↑) (Stall)	O.35(↑) (Visual)	O.46(↑) (Overall)
BOLA	$1.67_{\pm 0.66}$	$2.16_{\pm 0.57}$	$1.47_{\pm 0.27}$	$3.87_{\pm 0.69}$	$4.43_{\pm 0.61}$	$3.68_{\pm 0.71}$	$2.57_{\pm 1.27}$	$3.09_{\pm 1.26}$	$2.37_{\pm 1.20}$
RobustMPC	$2.01_{\pm 0.66}$	$2.14_{\pm 0.69}$	$1.56_{\pm 0.28}$	$3.97_{\pm 0.76}$	$4.40_{\pm 0.59}$	$3.74_{\pm 0.71}$	$2.81_{\pm 1.19}$	$3.07_{\pm 1.29}$	$2.45_{\pm 1.18}$
Pensieve	$1.44_{\pm 0.44}$	$2.18_{\pm 0.78}$	$1.43_{\pm 0.16}$	$3.32_{\pm 1.24}$	$4.71_{\pm 0.29}$	$3.44_{\pm 0.96}$	$2.21_{\pm 1.26}$	$3.21_{\pm 1.39}$	$2.25_{\pm 1.17}$
Fugu	$2.44_{\pm 0.85}$	$2.08_{\pm 0.66}$	$1.62_{\pm 0.32}$	$4.02_{\pm 0.61}$	$4.41_{\pm 0.69}$	$3.76_{\pm 0.70}$	$3.09_{\pm 1.09}$	$3.03_{\pm 1.33}$	$2.50_{\pm 1.17}$
Comyco	$2.24_{\pm 0.75}$	$2.29_{\pm 0.60}$	$1.72_{\pm 0.34}$	$3.44_{\pm 1.03}$	$4.58_{\pm 0.45}$	$3.47_{\pm 0.84}$	$2.73_{\pm 1.06}$	$3.34_{\pm 1.16}$	$2.43_{\pm 1.04}$
A²BR	$3.35_{\pm 1.03}$	$2.46_{\pm 0.28}$	$2.04_{\pm 0.44}$	$4.48_{\pm 0.20}$	$4.11_{\pm 0.54}$	$3.78_{\pm 0.47}$	$3.81_{\pm 0.98}$	$3.13_{\pm 0.91}$	$2.75_{\pm 0.97}$

2) Bitrate-Stall Analysis: We summarize the experimental results according to the relationship of stall time and normalized bitrate, in which the normalized bitrate is computed as $\frac{R_k}{160}$. Notably, results show that our proposed BOLA performs much better than the results of the previous work [56] (2% vs. 5% on stall ratio). Due to the minor incorrect settings in terms of the experimental setup [1], we have re-evaluated BOLA with tuned parameters to achieve the best *average* performance across all network traces.

Figure 10 reports the comparison results of existing ABRs, including heuristics and learning-based schemes. Recall that A^{2}BR is only tuned for 20-shot. We see that A^{2}BR can provide outstanding performance in terms of high video bitrate and low stall ratio. Specifically, Figure 10(a) demonstrates that A^{2}BR outperforms heuristics on the average bitrate of 2.9%-6% and average stall reduction of 42%-70%. One of the underlying reasons is that the bitrate ladder provided by the 4K video is not adequate for 4G networks. Commonly, the average bandwidth of 4G networks is lower than 40Mbps, while most bitrate levels are larger than the average bandwidth. To this end, how to construct a proper bitrate ladder for different network conditions is an interesting topic but out of scope here [58]. We will jointly consider the bitrate ladder construction and ABR implementation in future work.

Figure 10(b) and 10(c) show that A^{2}BR increases the bitrate by 2%-5% and heavily reduces the stall ratio by 6%-23% compared with state-of-the-art ABR Fugu [2] in the 4G scenario. Furthermore, Figure 10(d) shows a significant performance gap between the recent ABR scheme and A^{2}BR , since recent work suffers from either low video bitrate (e.g., BOLA and RobustMPC) or high stall ratio (e.g. Pensieve and Comyco). From Figure 10(f) and Figure 10(e), we find the same conclusion of prior work [56]: NN-based ABR scheme such as Pensieve and Comyco fails to maintain the high performance in 5G network scenario because they often suffer from very high stall ratio. In contrast, the aforementioned situation has been rectified by A^{2}BR : it shows a significant decrease (i.e., 69% and 80% on average) of video stall in

[1]See details in https://github.com/SIGCOMM21-5G/artifact/issues/8



(a) Modification to A^{2}BR to support QoE-aware learning.

(b) HSDPA dataset

(c) FCC dataset

Fig. 11. Comparing A^{2}BR with existing QoE-driven ABR algorithms on HSDPA and FCC networks traces. Results are normalized against the performance of Pensieve. The error bars show std from the average. We consider three types of QoE metrics that described in §V-E2.

driving and walking scenario compared to Pensieve, as the average bitrate of it only performs 2%-4% lower than that of Pensieve. Such decreases are indeed acceptable because each algorithm performs with the normalized bitrate of at least 0.9 in 5G scenarios.

3) ITU-T Rec P.1203 QoE Analysis: In addition, we calculate an estimated MOS for each video session via ITU-T P.1203 QoE model [55]. As suggested by previous work [59], we use mode 0, which fully considers 6 metrics, such as selected bitrate, video codec, video resolution, frame rate, starting time of stall events, and stall duration. During the evaluation phase, we record the detailed playback behavior for each chunk and ABR algorithm. Then we feed the playback logs to ITU-T P.1203 Standalone Implementation [2] and obtain the final MOS scores. To better understand the quality from different perspectives, we take 3 MOS scores, involving i) O.23, indicating perceptual stalling indication; ii) O.35, meaning visual coding quality score for the entire session; iii) O.46, representing media session quality score, aka. final QoE score. All the MOS scores are computed as a single score on a 1-5 quality scale. The comprehensive instruction of ITU-T P.1203 can be found in [19].

Table III reports the detailed comparison results of A^{2}BR and existing ABR algorithms over 4G and 5G network conditions, where the values are depicted as *avg.*$\pm$*std.*. There are three key takeaways from these results. First, we find that A^{2}BR either rivals or surpasses the performance of the best existing ABR algorithm on each MOS score and network considered. Especially, in comparison to the closest competitive scheme Fugu, A^{2}BR provides 10.0% (4G: 25.93%, 5G: 0.53%) on average O.46 score (i.e., overall QoE score). In our opinion, A^{2}BR obtains its outstanding performance since it pays more attention to avoiding stalls, as it improves 23.3%-72.4% on average O.23 score (i.e., overall stall score) compared with baselines.

Second, A^{2}BR not only reaches the best average performance but also attains the lowest variance with an average O.46 score compared to existing ABR schemes. Here the key reason is that A^{2}BR prefers smooth rate adaptation rather than requesting chunks with the highest bitrate level (e.g. Pensieve). Particularly, Comyco reaches second place in terms of QoE variation (1.04), but it only ranks fourth on average QoE (2.43) among 6 candidates. Compared to the best scheme A^{2}BR , Comyco should focus on reducing the stall ratio in 5G network conditions. A strawman solution is to set the rebuffering penalty as a larger value, e.g., μ=320. However, the increased rebuffering penalty also leads to the more conservative bitrate selection strategy on the 4G network. Thus, the better solution for Comyco is to follow A^{2}BR , i.e., learning policies with different network conditions separately (§III-B).

Finally, to our surprise, we observe that Pensieve performs poorly on O.46 score, even though it stands for the worst scheme among all ABR schemes. This is because Pensieve obtains the lowest score on O.23 (i.e., stall). Such results indicate that the distribution shift effect is positively misleading the algorithm to local optima [60]. A^{2}BR performs well over all networks considered since it can continually learn the strategy to vary current network environments.

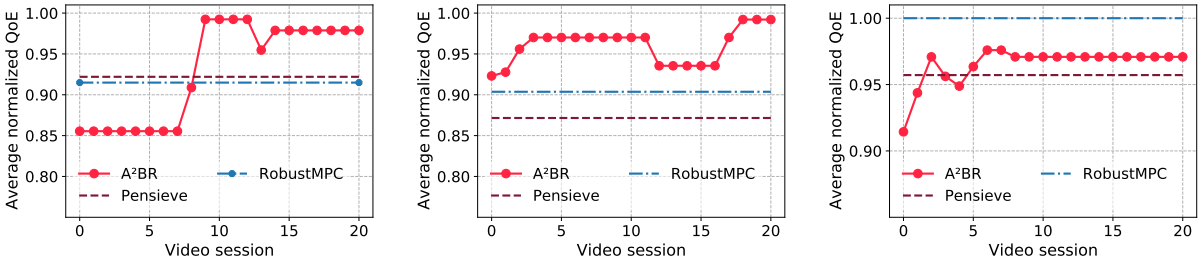
E. Case Study: Varying User Preference

Previous experiments assume that the QoE of a user is roughly even with the others [8], [11], [14], [2]. Based on the assumption, traditional ABRs are optimized toward a fixed QoE model, and these approaches will perform poorly if the QoE metric is changed [12]. However, recent studies argue that QoE preferences vary with users because each user has their viewing interests. Hence, the ABR algorithm should be optimized by considering QoE diversity. In this part, we try to answer: can A^{2}BR also tame the complexity of various QoE preferences in relatively few-shot?

1) Enhancement for multiple QoE preferences: We consider two enhancements to the training algorithm which enable A^{2}BR to better understand QoE preferences. The detailed modification is described in Figure 11(a). The first change is that we incorporate QoE parameters μ, σ into the state representation. During training, for each session, we randomly reset the parameters and obtain the reward score according to the selected parameters. Second, we apply a fully-connected layer with 64 neurons to the A^{2}BR 's NN for extracting the high-dimensional features from the QoE parameters.

2) Experimental settings: At the offline stage, we train A^{2}BR via Pensieve training dataset [62]. At the online stage, we continually train the meta-policy over the traces which

[2]https://github.com/itu-p1203/itu-p1203



(a) Emulation on Mahimahi [61]. (b) Real-world experiments: public WiFi. (c) Real-world experiments: wired network.

Fig. 12. Performance of A^{2}BR and recent ABRs in Dash.js. Results are reported as normalized QoE (QoE / QoE$_{max}$).

are randomly selected from the HSDPA and FCC datasets. Training time lasts 20-shot. In light of MPC's evaluation settings [8], we compare the performance of ABR algorithms under 3 sets of QoE weights: i) "Default" ($\mu = 4.3, \lambda = 1$), ii) "Conservative" ($\mu = 8.6, \lambda = 1$), and iii) "Stable" ($\mu = 4.3, \lambda = 4.3$). In this experiment, RobustMPC is optimized by the actual QoE preference. We adopt Pensieve as the pre-trained model that trained with the default QoE set.

3) Effectiveness analysis of A^{2}BR : Figure 11 provides a summary to illustrate the average normalized QoE that each scheme achieves on HSDPA and FCC datasets. As expected, A^{2}BR efficiently learns the individual QoE requirements and outperforms ABR approaches: it rapidly obtains 5.1% QoE gains over HSDPA and 5.0% QoE gains over FCC compared with A^{2}BR-meta (i.e., the initial meta-policy of A^{2}BR) in 20-shot. Moreover, another observation is that heuristics like RobustMPC fails to handle all QoE preference settings. Especially in the "Stable" QoE set, RobustMPC never picks the chunk with the highest bitrate during the entire session. We reason that RobustMPC solves a QoE maximization problem over a horizon of only *5* future chunks. While due to the large penalty from switching bitrates (i.e., 4.3), the best trajectory that MPC planned seldom contains any bitrate changes. Widen the prediction horizon can not perfectly solve the problem for MPC since its sensitivity to throughput prediction errors and the length of the optimization horizon [8], [11]. It proves that there's still plenty of room for improvement by developing outstanding planning/decision strategies for heuristics. One of the possible ways is to joint deep learning and model-based planning method for accurate decision [63]. In addition, we can see that Pensieve performs poorly except for the default QoE set ($\mu = 4.3, \sigma = 1.0$) because it is trained via a fixed QoE objective. Thus, Pensieve can not adjust itself to various objectives such as the "Conservative" and "Stable" sets.

F. Real-world Deployment

We establish a full-system implementation to evaluate A^{2}BR and other ABR approaches on Dash.js [4]. Specifically, the system consists of a video client, a video server, and an A^{2}BR decision server. For each chunk, the video client reports all the features like a state to the A^{2}BR decision server. The decision server then sends the bitrate level back to the client. The client requests the chunk from the video server with the bitrate level suggested by the decision server. When the session ends, the decision server starts to restore the network traces from previously collected states, then it virtually rolls out several trajectories using a virtual player, and updating the NN according to the "real" and "fake" trajectories (see §IV-C). The network condition is configured by Mahimahi [61] with the randomly selected traces from HSDPA dataset [20]. We adopt TCP-BBR [64] as the basic TCP congestion control algorithm and repeatedly replay the videos named *EnvivioDash3* [4] with all the evaluated ABR algorithms over all considered network conditions. We report the average "learning curve" of A^{2}BR for each video session. Figure 12(a) shows the performance of A^{2}BR and existing ABR schemes (i.e., Pensieve and RobustMPC) in the emulation environment, we see that A^{2}BR (i.e., meta policy) performs worse than Pensieve and RobustMPC at the beginning of the online stage. After playing the video 8 times, A^{2}BR suddenly learns the better policy that matches the behavior of baselines. After that, we see that A^{2}BR incrementally improves itself to achieve the best results among candidates, i.e., Pensieve: (7.8%↑) and RobustMPC: (8.5%↑).

Furthermore, we conduct a real-world experiment over two representative network scenarios, including public WiFi and wired network. We apply the video client and A^{2}BR server on the laptop (MacBook Pro, 64GB RAM), and establish a video server on the AWS. For the first experiment, we connect a public WiFi and repeatedly play the video 20 times. Results are shown in Figure 12(b). Notably, this scenario is a typical network condition over which the basic network "trace" is dynamically changed rather than fixed during the phase. Different from the results in the emulation tools, we find that Pensieve's conservative policy leads to poor generalization ability: it doesn't work well on networks dissimilar to the networks it has trained on [65], [2]. A^{2}BR , trained 20 times, outperforms other ABR algorithms on average QoE improvements of 9.6% (RobustMPC) and 12.8% (Pensieve), which yields the effectiveness and generalization capacity.

The second real-world experiment is established over the wired network, where the average bandwidth is always sufficient for picking the chunk with the highest bitrate. We demonstrate the results in Figure 12(c) and find that Pensieve still exhibits the worst behavior because it performs like running on slow-path networks. By contrast, RobustMPC runs perfectly well in the high throughput and low variance network conditions. Unfortunately, we can find that A^{2}BR

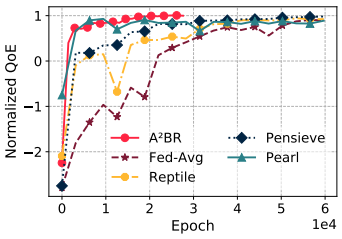


Fig. 13. Comparing A^{2}BR with existing meta-RL methods, including Pearl, Reptile and Fed-Avg.

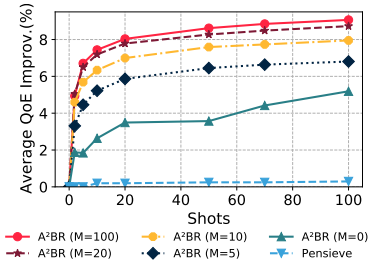


Fig. 14. A^{2}BR with different rollout times M.

	1-step MPC	NN
CPU(%)	$12.79_{\pm 0.94}$	$14.60_{\pm 1.11}$
Memory (MB)	$79.38_{\pm 0.8}$	$97.92_{\pm 0.85}$
Inference (ms)	-	$0.77_{\pm 0.91}$
Energy (/h)	12.68%	12.85%

Fig. 15. Comparing performance of 1-step MPC and NN-based ABR algorithm. The results contain CPU, memory consumed, inference time, and energy cost.

's performance is continually enhanced by 5.6% but fails to reach the optimal score after 20 trials. The reason is similar to the fact that we have figured out in §V-C4: A^{2}BR is incrementally improved by encouraging exploration, which might heavily influence the ABR performance.

TABLE IV
A^{2}BR WITH DIFFERENT ROLLOUT M

Infer. (KFLOPS)	M=5	10	20	100
11.3	123.9	208.4	377.4	1729.0

VI. Ablation Studies

A. Choice of Meta-learning Methods

We make a comparison of A^{2}BR with different meta-learning strategies, such as Pearl [66], Reptile [67] and FedAvg [68], and the vanilla RL training methodology [11]. Pearl is an off-policy meta-RL algorithm that takes probabilistic embeddings to determine the latent embedding of the current environment. Considering A^{2}BR is working in the discrete action spaces (§IV), we use double Q-learning [69] as the Pearl's RL training method. In the offline training stage, the latent embedding is directly estimated by the average and standard deviation throughput of the entire session. While in the online stage, the embedding is computed from the throughput observed of the past five chunks. Reptile is one of the meta-learning algorithms which repeatedly samples a task, training on it, and updating the initial parameters towards the parameters learned on that task. In addition, recent work reveals that FedAvg and Reptile are quite similar to each other since FedAvg can be viewed as a special case of Reptile if the learning rate equals 1 [70], [71]. Hence, we treat FedAvg as a linear combination of a naive baseline. We evaluate ABRs on the same validation set every 300 epochs and report the learning curve in Figure 13. A^{2}BR trains faster and performs slightly better than others. Comparing A^{2}BR with Pearl, we find Pearl typically converges quickly because Pearl works based on a sample-efficient off-policy method. However, it fails to obtain better final performance compared with on-policy meta-RL A^{2}BR . What's more, compared to Reptile, both the convergence speed and final performance of A^{2}BR are significantly improved, which indicates the effectiveness of the offline stage. To sum up, MAML is the most suitable method among existing meta-RL algorithms for our work.

B. Choice of Different Rollout M and Learning Epochs

The higher M indicates better sample efficiency since most trajectories will be sampled from the virtual player and the environment collector. However, it also brings out the risk that too many virtual trajectories may be overkill for meta updating. We compare A^{2}BR with different rollout times M, which includes $\{0, 5, 10, 20, 100\}$, over the Puffer dataset. Note that A^{2}BR doesn't use the virtual player for improving learning efficiency when $M = 0$. In other words, it reflects the performance of using MAML solely at the online stage. The results of training A^{2}BR in 100-shot are shown in Figure 14. We leave three notes here. First, A^{2}BR reaches the best performance when M=100. However, it takes 5× on computational overhead but only improves less than 1% on average QoE compared with M=20. Thus, we confirm that M=20 is a sufficient parameter setting for the online stage.

Second, compared to Pensieve which doesn't adopt meta-policy techniques, we see that A^{2}BR (M=20) consistently improves the performance by 6% in 10-shot, 7.8% in 20-shot, but only *8%* in 50-shot. Such minor improvements (i.e., 0.2%) between 20-shot and 50-shot motivates us to continually train A^{2}BR in 20-shot at the online stage.

Finally, we find that A^{2}BR with M=0 obtains 3.5% improvements on average QoE compared to Pensieve in 20-shot. It proves that A^{2}BR can also provide acceptable improvements without using a virtual player to replay the environment experienced. Unfortunately, it only gains 4% improvements compared with Pensieve in the next 50-shot, which is indeed a minor improvement compared with using virtual play technologies.

C. Computational Cost for Online Learning

We follow the calculation method described in [72] and compute the number of floating point operations (FLOPs) of A^{2}BR on both inference and backpropagation operation [73] in Table IV. A^{2}BR with M=100 takes almost 5× computational overhead compared with M=20. Hence, we set M=20 for balancing the trade-off among the sample efficiency, model accuracy, and computational cost. Most notably, this cost is rather small, only 0.86% of the consumption inferred by the state-of-the-art image recognition model MicroNet [74].

Moreover, we deploy the NN model of A^{2}BR to the mobile phones to investigate whether A^{2}BR can work well or not. In detail, we adopt an Android phone named Huawei P20, with 128GB of internal storage and 4GB of RAM. We modify Kuaishou's production video player to support NN inference via a self-developed NN tool, namely YCNN (a Tensorflow-Lite [75] like NN API). Figure 15 demonstrates the performance of two ABR schemes, i.e., 1-step MPC and NN-based scheme, in which these two schemes have been performed in the same video and network environment for at least two hours. The NN-based scheme uses the same architecture of A^{2}BR , and it only makes inference instead of continual learning. As shown, we can see the energy cost of the two schemes is quite similar. Meanwhile, the NN-based scheme runs slightly higher than 1-step MPC in terms of CPU utilization and memory consumed. Such minor costs have negligible impacts on today's mobile phones [76]. Moreover, the extra-low overhead on inference time proves that applying sophisticated feature engineering is useful for both meta-learning and online deployment.

VII. Related Work

In this section, we summarize recent ABR algorithms. Existing ABR algorithms are generally categorized into three types: heuristics, learning-based, online-learning based, and preference-aware ABR approaches.

Traditional ABRs: Heuristic-based ABR methods often adopt critical features or domain knowledge, such as throughput prediction (E.g., FESTIVE [6] and PANDA [77]) and buffer occupancy control (E.g., BBA [7] and BOLA [51]), for choosing the proper bitrate for the ABR task. However, such approaches require accurate bandwidth estimation or suffer from long-term bandwidth fluctuation problems. Then, MPC [8] picks the next chunks' bitrate by jointly considering throughput and buffer occupancy. Nevertheless, MPC is sensitive to its parameters since it relies on well-understanding different network conditions. To deal with the aforementioned shortcomings, Oboe [10] is an auto-tuning method to tune the traditional heuristic methods to achieve better performance in different network settings. Moreover, in the live streaming field, MultiLive designs a quality model and proposes a rate adaptation algorithm for multi-party scenarios [78]. However, such heuristics will perform unstable if the current network condition doesn't meet the presumptions of the fundamental principle of the proposed ABR algorithm.

Learning-based ABRs: By contrast, learning-based schemes take raw observations as the input, aiming to train a NN from the clean slate via various learning methods, such as imitation learning [14], A3C [11], PPO [23], and ACKTR [79]. For example, Mao et al. [11] propose Pensieve, which leverages the deep reinforcement learning (DRL) method to generate a strategy towards higher reward feedback, in which the reward function is represented as the simple weighted sum of bitrate, rebuffering, and smoothness. Bentaleb et al. [59] propose AMP that encompasses techniques for bandwidth prediction and model auto-selection, which is specifically designed for low-latency live steaming with chunked transfer encoding. Stick is a fusion approach that fuses the learning-based and the conventional buffer-based approach [12] for not only achieving higher performance but also reducing the computational overhead. Moreover, to make the learning-based ABR scheme more practical, Meng et al. [80] proposes Pitree to distill the ABR policy into a decision tree-based model. Meanwhile, Lumos [81] leverages the regression tree for accurate throughput prediction, leading to better QoE performance. Such approaches are trained or optimized in the offline setting, that is, using a fixed network distribution. Nevertheless, they fail to perform well if the online network distribution is different from the training set.

Online-learning based ABRs: Several online-learning-based ABR schemes have been proposed in recent years. OnRL [41] adopts federated learning to continually update its strategy for real-time communication. Oboe [10] and Puffer [2] dynamically update the configuration map or NN model according to current bandwidth capacities periodically. L2A-LL [82] uses Online Convex Optimization (OCO) to make decisions for low-latency live streaming. However, recent work failed to consider a personalized network environment with the fast adaptation requirement.

Preference-aware ABRs: Elephanta [83] is the first QoE diversity perception approach for edge clients by adjusting the parameters during the session. Elephanta models the video streaming process as a renewal system, which enables it to adapt to QoE diversity online. DAVS [84] is an imitation learning-based approach that considers the user's viewing preference for making the method adapt to the QoE diversity. Zuo et al. [85] propose Ruyi, an off-policy RL-based video streaming system that incorporates preference awareness into both the QoE model and the ABR algorithm. Ruyi is optimized by a variant of the Deep Q-learning algorithm with the experience replay technique [86]. However, recent schemes lack the ability to fast adapt to specific QoE preferences.

VIII. Conclusions and Future Work

We have proposed A^{2}BR , a novel meta-RL ABR approach to fast adapts the personalized network conditions. We divided the training process into two stages, aiming to meta-train an initial model in the offline stage, and continually leveraged domain knowledge to adapt tailor-made networks within few-shots in the online stage. Experimental results on several representative network scenarios revealed that A^{2}BR can quickly generate tailor-made policies within 20 shots.

In this work, we only discuss the performance of A^{2}BR in the VOD (video on demand) scenario, where the maximum buffer size is often set above 30 seconds. While another popular streaming scenario is the live streaming scenario, in which the buffer occupancy is considered as a penalty, often sized below 3 seconds. Ideally, deploying A^{2}BR in live (or low latency) streaming scenario is quite challenging, since i) measuring throughput becomes tough due to the application limit [87], ii) we should design the decision algorithm by considering lower playback buffer size [88], [89], and iii) the lack of a faithful *packet-level simulator* rather than frame-level solution [90] for live streaming. We plan to investigate A^{2}BR in the live streaming scenario in future work.

Furthermore, we also believe that A^{2}BR sheds light on improving similar input-driven MDP (IMDP) tasks, such as internet congestion control algorithms [33], scheduling/offloading algorithms [42], [91], and so on. In these tasks, an exogenous yet stochastic input process often affects the dynamics of the system as well. For example, for congestion control algorithms, heuristics like TCP-BBR and TCP-Cubic can't always perform well under all considered scenarios. Here A^{2}BR is a suitable scheme that allows the control strategy to adapt to the environment faster.

ACKNOWLEDGEMENT

We thank our colleagues from the Kuaishou video transport and delivery group, including Dan Yang, Yangchao Zhao, Yixuan Ban, and Kewei Zhu. We also thank the anonymous reviewer for the valuable feedback. Special thanks to our editors Prof. Zhu Han and Lei Liang for useful suggestions. This work was supported by the National Key R&D Program of China (No. 2018YFB1003703), NSFC under Grant 61936011, Beijing Key Lab of Networked Multimedia, and Kuaishou-Tsinghua Joint Project (No. 20192000456).

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

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

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

[5] 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.

[6] 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.

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

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

[9] 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.

[10] Z. Akhtar, Y. S. Nam, R. Govindan *et al.*, "Oboe: auto-tuning video abr algorithms to network conditions," in *SIGCOMM 2018*. ACM, 2018, pp. 44–58.

[11] H. Mao, R. Netravali, M. Alizadeh *et al.*, "Neural adaptive video streaming with pensieve," in *SIGCOMM 2017*. ACM, 2017, pp. 197–210.

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

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

[14] T. Huang, C. Zhou, R.-X. Zhang, C. Wu, X. Yao, and L. Sun, "Comyco: Quality-aware adaptive video streaming via imitation learning," in *Proceedings of the 27th ACM International Conference on Multimedia*. ACM, 2019, pp. 429–437.

[15] W. Li, J. Huang, S. Wang, S. Liu, and J. Wang, "Davs: Dynamic-chunk quality aware adaptive video streaming using apprenticeship learning," in *GLOBECOM 2020-2020 IEEE Global Communications Conference*. IEEE, 2020, pp. 1–6.

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

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

[18] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *International Conference on Machine Learning*. PMLR, 2017, pp. 1126–1135.

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

[20] 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.

[21] A. Narayanan, E. Ramadan, R. Mehta, X. Hu, Q. Liu, R. A. K. Fezeu, U. K. Dayalan, S. Verma, P. Ji, T. Li, F. Qian, and Z.-L. Zhang, "Lumos5g: Mapping and predicting commercial mmwave 5g throughput," in *Proceedings of the ACM Internet Measurement Conference*, ser. IMC '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 176–193. [Online]. Available: https://doi.org/10.1145/3419394.3423629

[22] 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.

[23] T. Huang, R.-X. Zhang, and L. Sun, "Self-play reinforcement learning for video transmission," in *Proceedings of the 30th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*, 2020, pp. 7–13.

[24] T. Huang, C. Zhou, X. Yao, R.-X. Zhang, C. Wu, B. Yu, and L. Sun, "Quality-aware neural adaptive video streaming with lifelong imitation learning," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 10, pp. 2324–2342, 2020.

[25] Y. Zheng, L. Lin, T. Zhang, H. Chen, Q. Duan, Y. Xu, and X. Wang, "Enabling robust drl-driven networking systems via teacher-student learning," *IEEE Journal on Selected Areas in Communications*, vol. 40, no. 1, pp. 376–392, 2021.

[26] H. Mao, S. B. Venkatakrishnan, M. Schwarzkopf, and M. Alizadeh, "Variance reduction for reinforcement learning in input-driven environments," *arXiv preprint arXiv:1807.02264*, 2018.

[27] M. T. Spaan, "Partially observable markov decision processes," in *Reinforcement Learning*. Springer, 2012, pp. 387–414.

[28] Z. Huo, Q. Yang, B. Gu, L. C. Huang *et al.*, "Faster on-device training using new federated momentum algorithm," *arXiv preprint arXiv:2002.02090*, 2020.

[29] X. Jiang, H. Wang, Y. Chen, Z. Wu, L. Wang, B. Zou, Y. Yang, Z. Cui, Y. Cai, T. Yu, C. Lv, and Z. Wu, "Mnn: A universal and efficient inference engine," in *MLSys*, 2020.

[30] T. Hospedales, A. Antoniou, P. Micaelli, and A. Storkey, "Meta-learning in neural networks: A survey," *arXiv preprint arXiv:2004.05439*, 2020.

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

[32] Z. Meng, M. Wang, J. Bai, M. Xu, H. Mao, and H. Hu, "Interpreting deep learning-based networking systems," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, 2020, pp. 154–171.

[33] S. Abbasloo, C.-Y. Yen, and H. J. Chao, "Classic meets modern: A pragmatic learning-based congestion control for the internet," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, 2020, pp. 632–647.

[34] J. Schulman, X. Chen, and P. Abbeel, "Equivalence between policy gradients and soft q-learning," *arXiv preprint arXiv:1704.06440*, 2017.

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

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

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

[38] A. Biswas, "First-order meta-learned initialization for faster adaptation in deep reinforcement learning," in *preprint 2018*, 2018.

[39] H. Zhang, A. Zhou, Y. Hu, C. Li, G. Wang, X. Zhang, H. Ma, L. Wu, A. Chen, and C. Wu, "Loki: improving long tail performance of learning-based real-time video adaptation by fusing rule-based models," in *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*, 2021, pp. 775–788.

[40] C. Liu, M. Xu, Y. Yang, and N. Geng, "Drl-or: Deep reinforcement learning-based online routing for multi-type service requirements," in *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*. IEEE, 2021, pp. 1–10.

[41] 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 *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, 2020, pp. 1–14.

[42] H. Mao, M. Schwarzkopf, S. B. Venkatakrishnan, Z. Meng, and M. Alizadeh, "Learning scheduling algorithms for data processing clusters," in *Proceedings of the ACM Special Interest Group on Data Communication*, 2019, pp. 270–288.

[43] H. Mao, M. Schwarzkopf, H. He, and M. Alizadeh, "Towards safe online reinforcement learning in computer systems," in *33rd conference on neural information processing systems (NeurIPS 2019)*, 2019.

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

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

[46] 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.

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

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

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

[50] 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*, 2018, pp. 43–48.

[51] 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.

[52] 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.

[53] T. Huang, "Oboe reproduce," https://github.com/godka/oboe-reproduce, 2022.

[54] J. J. Quinlan and C. J. Sreenan, "Multi-profile ultra high definition (uhd) avc and hevc 4k dash datasets," in *Proceedings of the 9th ACM Multimedia Systems Conference*, 2018, pp. 375–380.

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

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

[57] T. H. Channel, "Real 4k hdr 60fps: Lg jazz hdr uhd (chromecast ultra)," https://www.youtube.com/watch?v=mkggXE5e2yk, 2021.

[58] T. Huang, R.-X. Zhang, and L. Sun, "Deep reinforced bitrate ladders for adaptive video streaming," in *Proceedings of the 31st ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*, 2021, pp. 66–73.

[59] A. Bentaleb, A. C. Begen, S. Harous, and R. Zimmermann, "Data-driven bandwidth prediction models and automated model selection for low latency," *IEEE Transactions on Multimedia*, 2020.

[60] A. Agarwal, S. M. Kakade, J. D. Lee, and G. Mahajan, "On the theory of policy gradient methods: Optimality, approximation, and distribution shift," *Journal of Machine Learning Research*, vol. 22, no. 98, pp. 1–76, 2021.

[61] 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.

[62] H. Mao, "Pensieve-traces," Jul 2017. [Online]. Available: https://www.dropbox.com/sh/ss0zs1lc4cklu3u/AAB-8WC3cHD4PTtYT0E4M19Ja?dl=0

[63] T. Huang and L. Sun, "Deepmpc: A mixture abr approach via deep learning and mpc," in *2020 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2020, pp. 1231–1235.

[64] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "Bbr: Congestion-based congestion control," *Queue*, vol. 14, no. 5, p. 50, 2016.

[65] P. Crews and H. Ayers, "Cs 244'18: Recreating and extending pensieve, 2018," 2018.

[66] K. Rakelly, A. Zhou, C. Finn, S. Levine, and D. Quillen, "Efficient off-policy meta-reinforcement learning via probabilistic context variables," in *International conference on machine learning*. PMLR, 2019, pp. 5331–5340.

[67] A. Nichol, J. Achiam, and J. Schulman, "On first-order meta-learning algorithms," *arXiv preprint arXiv:1803.02999*, 2018.

[68] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial Intelligence and Statistics*. PMLR, 2017, pp. 1273–1282.

[69] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, no. 1, 2016.

[70] Y. Jiang, J. Konečnỳ, K. Rush, and S. Kannan, "Improving federated learning personalization via model agnostic meta learning," *arXiv preprint arXiv:1909.12488*, 2019.

[71] M. Khodak, M.-F. F. Balcan, and A. S. Talwalkar, "Adaptive gradient-based meta-learning methods," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[72] OpenAI, "Ai and compute." 2018. [Online]. Available: https://openai.com/blog/ai-and-compute/

[73] 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.

[74] Y. Li, Y. Chen, X. Dai, D. Chen, M. Liu, L. Yuan, Z. Liu, L. Zhang, and N. Vasconcelos, "Micronet: Towards image recognition with extremely low flops," *arXiv preprint arXiv:2011.12289*, 2020.

[75] S. Li, "Tensorflow lite: On-device machine learning framework," *Journal of Computer Research and Development*, vol. 57, no. 9, p. 1839, 2020.

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

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

[78] Z. Wang, Y. Cui, X. Hu, X. Wang, W. T. Ooi, Z. Cao, and Y. Li, "Multilive: Adaptive bitrate control for low-delay multi-party interactive live streaming," *IEEE/ACM Transactions on Networking*, 2021.

[79] T. Feng, H. Sun, Q. Qi, J. Wang, and J. Liao, "Vabis: Video adaptation bitrate system for time-critical live streaming," *IEEE Transactions on Multimedia*, vol. 22, no. 11, pp. 2963–2976, 2019.

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

[81] G. Lv, W. Qinghua, W. Wang, Z. Li, and G. Xie, "Lumos: towards better video streaming qoe through accurate throughput prediction," in *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*. IEEE, 2021, pp. 1–10.

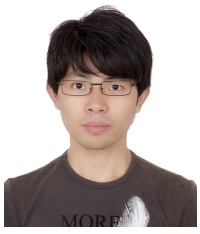
[82] T. Karagkioules, R. Mekuria, D. Griffioen, and A. Wagenaar, "Online learning for low-latency adaptive streaming," in *Proceedings of the 11th ACM Multimedia Systems Conference*, 2020, pp. 315–320.

[83] C. Qiao, J. Wang, and Y. Liu, "Beyond qoe: Diversity adaptation in video streaming at the edge," *IEEE/ACM Transactions on Networking*, vol. 29, no. 1, pp. 289–302, 2020.

[84] W. Li, J. Huang, S. Wang, C. Wu, S. Liu, and J. Wang, "An apprenticeship learning approach for adaptive video streaming based on chunk quality and user preference," *IEEE Transactions on Multimedia*, 2022.

[85] X. Zuo, Y. Jiayu, M. Wang, and Y. Cui, "Adaptive bitrate with user-level qoe preference for video streaming," in *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*. IEEE, 2021, pp. 1–10.
[86] 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.
[87] A. Bentaleb, C. Timmerer, A. C. Begen, and R. Zimmermann, "Performance analysis of acte: A bandwidth prediction method for low-latency chunked streaming," *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, vol. 16, no. 2s, pp. 1–24, 2020.
[88] M. Lim, M. N. Akcay, A. Bentaleb, A. C. Begen, and R. Zimmermann, "When they go high, we go low: low-latency live streaming in dash. js with lol," in *Proceedings of the 11th ACM Multimedia Systems Conference*, 2020, pp. 321–326.
[89] L. Sun, T. Zong, S. Wang, Y. Liu, and Y. Wang, "Tightrope walking in low-latency live streaming: optimal joint adaptation of video rate and playback speed," in *Proceedings of the 12th ACM Multimedia Systems Conference*, 2021, pp. 200–213.
[90] G. Yi, D. Yang, A. Bentaleb, W. Li, Y. Li, K. Zheng, J. Liu, W. T. Ooi, and Y. Cui, "The acm multimedia 2019 live video streaming grand challenge," in *Proceedings of the 27th ACM International Conference on Multimedia*, 2019, pp. 2622–2626.
[91] R.-X. Zhang, T. Huang, M. Ma, H. Pang, X. Yao, C. Wu, and L. Sun, "Enhancing the crowdsourced live streaming: a deep reinforcement learning approach," in *Proceedings of the 29th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*, 2019, pp. 55–60.

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. He received Best Student Paper Awards presented by ACM Multimedia System 2019 Workshop.



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.

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.

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.

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