Meta Reinforcement Learning for Rate Adaptation

Abdelhak Bentaleb*¶, May Lim§, Mehmet N. Akcay‡, Ali C. Begen‡, and Roger Zimmermann§
*Concordia University, §National University of Singapore, ‡Ozyegin University
¶Corresponding Author: abdelhak.bentaleb@concordia.ca

IEEE INFOCOM 2023 - IEEE Conference on Computer Communications | 979-8-3503-3414-2/23/$31.00 ©2023 IEEE | DOI: 10.1109/INFOCOM53939.2023.10228951

***Abstract*—Adaptive bitrate (ABR) schemes enable streaming clients to adapt to time-varying network/device conditions to achieve a stall-free viewing experience. Most ABR schemes use manually tuned heuristics or learning-based methods. Heuristics are easy to implement but do not always perform well, whereas learning-based methods generally perform well but are difficult to deploy on low-resource devices. To make the most out of both worlds, we develop `Ahaggar`, a learning-based scheme running on the server side that provides quality-aware bitrate guidance to streaming clients running their own heuristics. `Ahaggar`'s novelty is the meta reinforcement learning approach taking network conditions, clients' statuses and device resolutions, and streamed content as input features to perform bitrate guidance. `Ahaggar` uses the new Common Media Client/Server Data (CMCD/SD) protocols to exchange the necessary metadata between the servers and clients. Experiments on an open-source system show that `Ahaggar` adapts to unseen conditions fast and outperforms its competitors in several viewer experience metrics.**

I. Introduction

With the prevalence of HTTP adaptive streaming (HAS), the design of adaptive bitrate (ABR) logic—the algorithm deciding which segments to download and when (primarily based on the advertised encoding bitrate)—has received significant research attention. Existing ABR schemes [14] can be broadly classified as heuristic or learning-based. ABR schemes driven by heuristics make decisions based on client-side observations such as throughput estimation [29], playback buffer level [45] or a combination of the two [51]. Although these schemes are easy to implement, they heavily depend on some configuration parameters and a poor setting may significantly hinder their efficacy [25]. Hence, learning-based schemes have become an alternative, benefiting from the latest breakthroughs in machine learning (ML) such as deep reinforcement learning (DRL), supervised and imitation learning techniques [7]. Learning-based schemes attain good strategies without requiring any presumptions about the environment.

Nonetheless, learning-based schemes are exposed to two major limitations. First, their performance heavily depends on the training data. Network environments can be quite diverse and their dynamics change over time. Therefore, future states are not easy to predict accurately. Most schemes use classical approaches to train an agent by giving it feedback for decisions while interacting with an environment. Such interaction can be efficiently performed in a controlled trace-driven simulator. Still, a mismatch may occur when the trained model is deployed in a live system and encounters an environment that was not previously seen [53]. As a result, the scheme may fail to perform proper rate adaptation. Second, deploying learning-based schemes on devices with scarce resources is impractical due to high storage and computational cost. Prior work [53] showed that a learning model trained on past network scenarios could hardly provide a comparable performance under new conditions, and hence, effective and continual model retraining/update was required. Last but not least, many studies [28], [20] claim that perceptual video quality and device resolution must be considered in the ABR logic to improve the quality of experience (QoE). Incorporating these parameters into a learning model and then continually retraining the model is also infeasible for clients running on low-resource devices.

A. Problem and Motivation

To confirm these claims, we performed some experiments. The test setup consisted of a DASH client [19] with five ABR schemes: (i) heuristic-based: BOLA (buffer-based) [45], Dynamic (buffer + throughput) [19] and RobustMPC [51]; (ii) learning-based: Pensieve [33] and Fugu [50] (trained on the network traces provided in their original papers). To emulate real-world network conditions, we used three 4G LTE network traces, selected randomly from [34], [49] with different bandwidth fluctuation patterns ⟨average, standard deviation⟩, and user mobility types, namely: Metro ⟨1.5,0.5⟩ Mbps, Bus ⟨5.6,2.3⟩ Mbps, and Car ⟨4.3,2.5⟩ Mbps. For the video content, we used Akamai's *EnvivioDash3* [5], which is 180 seconds long, consisting of 90 two-second segments encoded with H.264 using the following ABR ladder: {0.3Mbps@180p, 0.75Mbps@360p, 1.2Mbps@480p, 1.8Mbps@576p, 2.8Mbps@720p, 4.3Mbps@1080p}. We kept the dash.js default max buffer size of 20 seconds and used VMAF [40] to measure the quality. The streaming client ran on a Samsung Galaxy S9 mobile phone with 64 GB of internal storage, Android v.10 OS, 4x2.8 GHz Kryo 385 CPU, Qualcomm Adreno 630 GPU and 3000 mAh (3.8V) battery.

Table I summarizes the average QoE results from over ten runs (bitrate, quality and total rebuffering duration) and resource usage metrics (CPU, memory and energy consumption). We measured resource usage metrics using device tools, including the Simple System Monitor, Android Profiler and Battery Manager. The energy consumption includes video rendering, communication with the server and ABR decisions (*i.e.*, mathematical rule or model inference). We used the TensorFlow Lite and Converter toolkits [23] to deploy the Fugu and Pensieve models on mobile devices. In conclusion, heuristic-based schemes always use the least resources but deliver the best QoE metrics only for 4G LTE (Metro). This confirms that heuristic-based schemes are practical but fail to perform equally over diverse network conditions. Also, learning-based schemes achieve better performance in environments seen during the training and can be partially generalized for other conditions, as shown in the results of 4G LTE (Bus) and (Car), but they consume more resources and energy.

TABLE I: Average QoE results and resource usage metrics for different ABR schemes using three 4G LTE network conditions (↑: higher is better (green), ↓: lower is better (green), lowest performance (red), ±: std).

ABR	Bitrate & Quality (Mbps & VMAF)	Rebuffering Duration (s)	CPU Usage (%)	Memory (MB)	Energy (kJ)
4G LTE (Metro)					
BOLA	**1.20 & 64.8** ↑	**1.77** ↓	09.11±0.22	66.10±0.56	3.80
Dynamic	1.08 & 61.6	1.90	**08.75±0.15** ↓	**60.04±0.10** ↓	**3.40** ↓
RobustMPC	1.00 & 60.7	1.82	12.80±0.96	75.11±0.77	3.90
Pensieve	0.95 & 59.2	2.10	19.60±1.22	98.98±0.85	6.10
Fugu	1.06 & 61.3	2.00	17.22±1.05	97.93±0.73	6.03
4G LTE (Bus)					
BOLA	1.94 & 71.22	1.99	**10.22±0.32** ↓	**68.19±0.45** ↓	**3.83** ↓
Dynamic	2.11 & 74.43	2.13	10.78±0.37	68.44±0.47	3.85
RobustMPC	2.38 & 76.33	1.94	14.22±1.01	81.15±0.66	5.22
Pensieve	2.94 & 82.79	1.23	20.55±1.66	101.2±1.11	8.77
Fugu	**3.10 & 83.15** ↑	**1.12** ↓	22.43±1.89	105.3±0.94	8.91
4G LTE (Car)					
BOLA	1.60 & 67.42	6.66	09.25±0.20	66.88±0.60	3.82
Dynamic	1.17 & 63.18	3.34	**08.81±0.18** ↓	**61.19±0.11** ↓	**3.50** ↓
RobustMPC	1.66 & 67.80	4.10	13.41±1.05	77.12±0.84	4.24
Pensieve	**2.33 & 75.88** ↑	**1.42** ↓	20.36±1.74	100.1±1.44	7.36
Fugu	2.30 & 74.22	1.54	20.07±1.33	99.97±0.99	7.30

We argue that heuristic and learning-based schemes can complement each other and leveraging the advantages of both solutions while avoiding their shortcomings is the key. This brings up the following three questions, which we seek to answer: ❶ Can we run a lightweight heuristic-based scheme on the client side and a learning-based bitrate guidance on the server side (which is not as constrained as the clients) such that they can cooperate harmoniously to deliver s better QoE? ❷ How to implement bitrate guidance with perceptual quality and device resolution awareness? ❸ How to achieve continual learning for the server-side bitrate guidance?

B. Ahaggar: A Meta-RL Approach for ABR

We answer the questions above in the context of Ahaggar[1], a meta reinforcement learning (meta-RL)-based solution. Ahaggar has a server-side learning model that takes network conditions, clients' statuses, device resolutions and streamed content as input features, and then provides quality and resolution-aware bitrate guidance to the streaming clients. Leveraging the server's vast computational power, storage capacity and memory, Ahaggar enables model inference for performing bitrate guidance tasks and helps resource-constrained streaming clients run their lightweight heuristic-based ABR schemes. Ahaggar models bitrate guidance tasks for multiple clients as a partially observable Markov decision process (POMDP) and leverages the latest developments in DRL to dynamically adapt to the varying network conditions. Specifically, it uses advantage Actor-Critic networks (A2C) for model training and Distributed Proximal Policy Optimization (DPPO) [24] with clip and Adam optimizer for policy updates at each time interval. Considering the changes in the environment, we adopt a Model Agnostic Meta-Learning (MAML) [21] on-policy gradient-based meta-RL approach that embeds policy gradient steps into the meta optimization. This allows Ahaggar to update the model parameters to achieve good generalization performance on unseen environments during the inference. Therefore, our model can converge quickly to the best performance and adapt to new unseen environments with only a small number of (*e.g.*, 40) shots. To our knowledge, this paper is the first study using meta-RL to improve QoE for adaptive streaming clients while cleanly separating the responsibilities for the servers and clients and respecting the client-driven nature of HAS.

The Ahaggar solution comprises two phases: (i) (offline) meta-training where each RL agent trains the Ahaggar meta-model on heterogeneous network environments, and (ii) (online) meta-testing (also called inference) where each agent continually learns the system dynamics and rapidly optimizes the meta-policy, adjusting the parameter weights that determine the agent behavior according to the trajectories collected from both the meta-training and meta-testing. We take inputs from the network, clients and streamed content into the Ahaggar neural network (NN) for bitrate guidance. The objective of Ahaggar is to select the minimum bitrate (among the available options) above which the next higher bitrate improves the perceptual quality only insignificantly at the specific device resolution. In this study, we use an objective full-reference perceptual video quality metric known as Video Multi-method Assessment Fusion (VMAF) [40].

To ensure healthy cooperation without incurring additional complexities between the clients and servers, Ahaggar adopts the emerging Common Media Client/Server Data standards: CMCD [17], [10], [13] and CMSD [18], [31]. CMCD defines a set of information collected by a media client and sent along with the HTTP requests to the server running Ahaggar in the form of query arguments or header extensions. CMSD allows the server to convey Ahaggar bitrate guidance decisions to media clients through the HTTP response headers.

We evaluate the performance of Ahaggar against several ABR solutions by running real-world trace-driven experiments. These experiments cover multiple clients with heterogeneous network conditions and device resolutions. Experimental results show that Ahaggar delivers consistent quality, improves viewer QoE by up to 87.0%, reduces rebuffering duration by up to 84.4% and reduces bandwidth consumption by up to 62.6%. In addition, Ahaggar quickly converges to the best solution during the training process with an improvement of 5.6× in terms of the number of epochs required and 6× speedup on the training time compared to the recent RL-based solutions such as [33], [50].

The contributions of this paper are three-fold:

1) Ahaggar is a learning-based, quality and resolution-aware bitrate guidance solution for HAS systems. It runs on the server and uses CMCD/SD to simplify the communication with the streaming clients.
2) Ahaggar models bitrate guidance tasks for multiple clients as a POMDP and uses an A2C (NN) architecture with clipped DPPO and Adam optimizer for policy updates, and meta-RL (MAML) for rapid and continual learning. As a result, Ahaggar learns new policies and adapts quickly.
3) A full system including the modified dash.js reference client [19] and Node.js with an HTTP server and

[1]A highland region in the central Sahara in southern Algeria.

Node JavaScript module (NJS application) implementing Ahaggar is publicly available at [3].

The rest of the paper is organized as follows. Section II shows the existing solutions for QoE optimization. Section III describes the Ahaggar solution, followed by the performance evaluation in Section IV. Section V concludes the paper.

II. Related Work

Client-Driven Heuristic-Based ABR: These schemes use heuristics based on estimated throughput (*e.g.*, PANDA [29]), buffer level (*e.g.*, BOLA [45]), segment size (*e.g.*, SARA [9]), or a combination (*e.g.*, MPCDASH [51]).

Client-Driven Learning-Based ABR: These schemes learn from the streaming environment by training an NN using DRL techniques [8], [16]. Mao *et al.* [33] proposed Pensieve, the first learning ABR that used DRL to generate a strategy toward maximizing the viewer QoE. Bentaleb *et al.* [11] designed AMP that implemented a set of learning-based bandwidth predictors and model auto-selection for HAS. Similarly, Fugu [50] was proposed to leverage the hidden Markov model for accurate throughput prediction. Huang *et al.* [25] used imitation learning to propose Comyco as ABR for on-demand videos.

Server-Driven Solutions: These solutions implement a rate control on the server to control a client's ABR decisions implicitly or explicitly. In the implicit control, the server does not require cooperation from the client. To that end, some solutions leveraged traffic shaping [6], [55] or super-resolution [27]. In the explicit control, the server receives information from the clients for intelligent QoE optimization decisions (*e.g.*, [13]).

Network-Driven Solutions: These solutions can be further categorized into: (1) *In-network solutions* where some works [12] use software-defined networking to assist clients in their ABR decisions, rate allocation [36] or multi-path delivery [15]; (2) *Server and network assistance solutions* where some papers [47], [38] leverage the SAND standard [2] that enables data collection from various network entities involved in media delivery. These data are then stored on a centralized server for intelligent decisions, *e.g.*, rate allocation; (3) *Data-driven solutions* that combine SAND with AI capabilities for improved decision making. These solutions collect QoE metrics from many streaming sessions at a logically centralized controller that maintains a global view of the real-time network conditions, based on which the controller makes decisions regarding the individual sessions (*e.g.*, [26], [22]).

III. Ahaggar Bitrate Guidance

Ahaggar serves *multiple clients* (agents in RL) with a *shared environment*, *distinct rewards* and *policies*, as depicted in Fig. 1. It performs bitrate guidance tasks at every time window and decides the best bitrate for each client. Therefore, we consider a fully cooperative multi-agent RL (MARL) [54] framework with *independent learners setting* that involves a set of agents sharing the same environment. In particular, we use a centralized training with decentralized execution

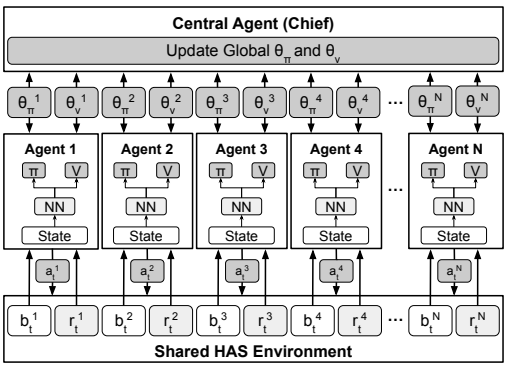


Fig. 1: MARL of Ahaggar.

(CTDE) paradigm [54] to train the MARL agents. CTDE allows these agents to train decentralized policies with global information during training and to make decisions based on the individually learned policies during inference. We also use MAML [21], the meta-RL algorithm, to adapt to various network environments through parameter learning. The overall workflow of Ahaggar is shown in Fig. 2, where the steps are numbered as ①–⑧.

A. Formulation of the Problem

At each segment download time epoch t, Ahaggar performs the bitrate guidance tasks (denoted by $\mathcal{Z}$) by selecting the best bitrate (denoted by l_t^c) with respect to the current state (denoted by s_t^c) of each client c. Mathematically, the bitrate guidance problem for multiple clients can be formulated as:

$$\begin{cases} \text{find } l_t^{c,\star}(\pi), \forall c \in [1,\dots,N], \forall t \in [1,\dots,k] \\ \arg\max_{\pi} QoE_t^c(\pi) \\ \text{s.t.} \quad l_t^{c,\star}(\pi) \le mtp_t^c \qquad \textbf{C.1} \\ \quad \sum_{c=1}^{N} l_t^{c,\star}(\pi) \le BW_{total} \quad \textbf{C.2} \end{cases} \tag{1}$$

where $l_t^{c,\star}$ is the best bitrate, which is the minimum among the available options above which the next higher bitrate improves the perceptual quality only insignificantly for the specific content at the specific device resolution. Here, we use 1-JND (Just Noticeable Difference) as the threshold for being significant [35]. Further in this formulation, π is an RL policy that decides the bitrate for each client, N is the total number of clients, BW_{total} is the total server capacity and mtp^c is the measured throughput by client c.

The formulation in (1) is a multi-agent decision problem and aims to find the best bitrate $l_t^{c,\star}$ that maximizes the viewer QoE_t^c for each client c with respect to **C.1–C.2**. Here, each client has access only to its local observations, and fully capturing the state of the global environment experienced by all clients is not feasible. Therefore, we cast the problem (1) as a partially observable Markov decision process (POMDP), which is characterized by its observation and historical information capabilities. The POMDP model consists of 11-tuples POMDP = ($\mathcal{S}$, $\mathcal{A}$, $\mathcal{O}$, $\mathcal{R}$, $\mathcal{P}$, $\mathcal{U}$, $\mathcal{Z}$, $\mathcal{C}$, N, α, γ), where:

- $\mathcal{S} = \{S^1, \dots, S^N\}$ is the set of the finite and discrete agent states of N agents. For each agent c, we define the set of agent states as $S^c = \{s_1^c, \dots, s_k^c\}$, where $k = |Z^c|$ is the total number of bitrate guidance tasks.

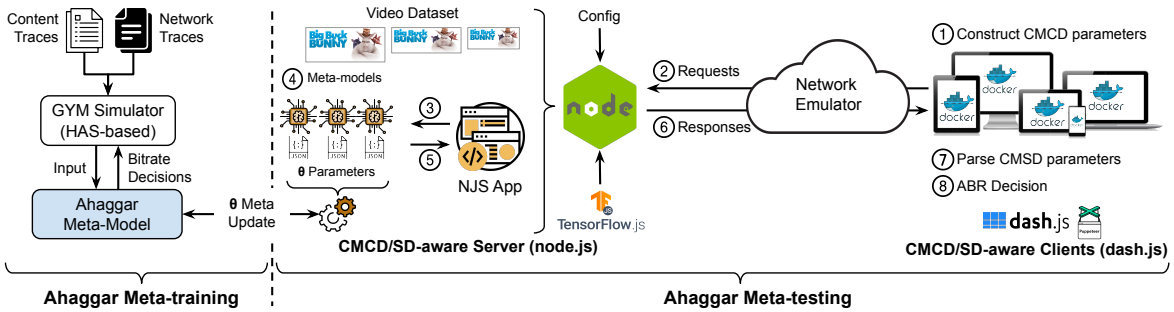


Fig. 2: The overall bitrate guidance system of `Ahaggar`.

- $\mathcal{A} = \{A^1, \ldots, A^N\}$ is the finite and discrete set of actions of N agents. For each agent c, we define the set of agent actions as $A^c = \{a_1^c, \ldots, a_k^c\}$, where each action is the selected bitrate l^c during a bitrate guidance task.
- $\mathcal{O} = \{O^1, \ldots, O^N\}$ is the finite set of observation states that are captured by the set of agents. For each client c, the set of observations is $O^c = \{o_1^c, \ldots, o_k^c\}$.
- $\mathcal{R} = \{R^1, \ldots, R^N\}$ is the set of expected immediate rewards, which depends on states and actions taken by N agents. For each client c, the set of rewards is $R^c = \{r_1^c, \ldots, r_k^c\}$.
- $\mathcal{P} = \mathcal{S} \times \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$ is the state transition probability function $\mathcal{P}(s'|s, a)$ from the state s to $s' \in S$ when action $a \in A$ is taken.
- $\mathcal{U} = \mathcal{O} \times \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$ is the observation probability function $O(o'|s', a)$ of observing $o' \in O$ after transitioning to s' due to a.
- $\mathcal{Z} = \{Z^1, \ldots, Z^N\}$ represents the bitrate guidance problem $\max_{\pi} QoE_t^c(\pi)$ for every agent c. The set of bitrate guidance tasks for agent c is thus defined as $Z^c = \{z_1^c, \ldots, z_k^c\}$.
- $\mathcal{C} = \{1, \ldots, N\}$ is the set of N agents, where N is the total number of agents and $c \in [1, \ldots, N]$ is an agent.
- α and $\gamma \in [0, 1]$ are the learning rate and discount factor, respectively.

At each time $t = [1, \ldots, k]$, each agent c does not track the exact state s_t^c, but rather it uses the observations o_t^c for any given task z_t^c. Therefore, it has to rely on the history of actions and observations, denoted by h_t^c, to perform the best actions that result in higher rewards. We define the set of histories of client c as $H^c = \{h_1^c, \ldots, h_k^c\}$ where $h_t^c = \{(a_t^c, o_t^c); \ldots; (a_1^c, o_1^c)\}$ and the set of histories of N agents as $\mathcal{H} = \{H^1, \ldots, H^N\}$. Yet, h_t^c might exponentially grow with every action taken and every state observed. In this case, the agent rather selects to use the belief states, denoted by B^c, which are single-valued and represent the observation probability U^c over all possible histories H^c in a given bitrate guidance task. For each $b_t^c \in B^c$, the observation probability distribution is denoted by $u_t^c = O(o_t^c|h_t^c, a_t^c)$, such that $O(o_t^c|h_t^c, a_t^c) = \sum_{s_{t+1}^c} \sum_{s_t^c} P(s_t^c|h_t^c)\, P(s_{t+1}^c|s_t^c, a_t^c)\, O(o_{t+1}^c|s_{t+1}^c, a_t^c)$, where $P(s_t^c|h_t^c)$ is the belief state b_t^c about the state s_t^c, $B^c = \{b_1^c, \ldots, b_k^c\}$ and $\mathcal{B} = \{B^1, \ldots, B^N\}$ are the set of belief states of agent c and the set of the finite and discrete belief states of N agents, respectively. These belief states are a sufficient measure of histories and given a belief state b_t^c, an agent c strives to find the effective optimal policy $\pi^{c,\star}$ to solve (1) by finding the best bitrate for each client that maximizes the accumulated discounted reward (denoted by G_t^c and defined in Section III-B).

The `Ahaggar` learning model solves the POMDP problem (1) using a multi-agent A2C [8] NN with clipped DPPO [24] and Adam optimizer for policy (π) updates at every time interval. For continual learning and quickly adapting to unseen environments, it uses MAML—the meta-RL policy gradient approach—allowing `Ahaggar` to learn hyper-parameter initialization and speed up the optimization of the learned model during inference.

B. `Ahaggar` Meta-Training (Offline)

To train the `Ahaggar` meta-model, we use *Park* [32]—a Python-based segment-level simulator that is based on OpenAI and state-of-the-art ABR simulators [44] for RL-based model training. This simulator faithfully emulates a streaming session in which the learning agent explores the streaming environment utilizing a large corpus of real-world network and content traces.

▷ **Network Traces.** We used the Belgium 4G/LTE [49], Norway 4G/LTE [41], NYU LTE [34] and Lumous 4G/5G [37] datasets. Each trace entry consists of a throughput value (Mbps), round-trip time (RTT; ms) and packet loss (%).

▷ **Content Traces.** We used the Comyco [25] and Waterloo SQoE-IV [20] datasets. To cover a wide range of device resolutions, each source video was encoded at {0.24, 0.37, 0.57, 0.75, 1.0, 1.76, 2.36, 3.0, 4.3, 5.7, 8.0, 11, 16.6}Mbps at a resolution of {180, 216, 288, 288, 360, 540, 720, 720, 1080, 1080, 1440, 2160, 2160}p, respectively. Each trace is comprised of video segments with their corresponding encoded bitrates (Mbps), sizes (byte) and VMAF scores for three device resolutions (phone, HDTV and UHDTV).

We performed customized modifications on the Park simulator to fully comply with the `Ahaggar` design. Specifically, we revised (i) the problem space using POMDP instead of MDP, (ii) input state, action and reward spaces, (iii) NN architecture with policy update and meta-RL approaches, (iv) headless video client by introducing three device resolutions, and (v) MARL with CTDE and shared environment support. During the session, the simulator used the traces and each client interaction with the environment as input features to feed into the NN, from which the RL agent in turn decided the segment bitrates at every time step.

The `Ahaggar` uses an A2C NN. Without loss of generality and since the agents are independent, we simplify the formulation in the context of a single agent. At every time epoch t, the segment-level statistics for each agent are collected and aggregated as the environment input state. Different from MDP, in POMDP, the agent cannot directly observe the complete

system state, but the agent makes observations that depend on the state. The agent uses these observations to form a belief about what state the system is currently in. This is called a belief state and is expressed as a probability distribution over all possible states. The solution of the POMDP is a policy prescribing which action to take in each belief state. Formally, RL agents interact with the environment that defines state space $\mathcal{S}$, observation space $\mathcal{O}$ and belief state space $\mathcal{B}$. At each time epoch t, each RL agent c observes a state $o_t^c \in \mathcal{O}$ and then receives a belief state $b_t^c \in \mathcal{B}$ from the environment. Later, it takes an action $a_t^c \in \mathcal{A}$ (*aka* $l_t^{c,\star}$) while it receives a reward $r_t^c \in \mathcal{R}$. Here, each agent c aims to find the optimal policy $\pi^{c,\star} : \mathcal{S} \rightarrow \mathcal{O} \rightarrow \mathcal{B} \rightarrow \mathcal{A}$ that maps states-to-actions and maximizes the reward.

▷ **Input State.** At each time epoch t, each agent c takes a belief state with inputs defined as $b_t^c = \{mtp_t^c, qt_t^c, bl_t^c, ls_t^c, dt_t^c, rs_t^c, \overrightarrow{LS_t^c}, \overrightarrow{QT_t^c}\}$, comprised of network, content and playback features of the last downloaded segment. These inputs are: measured throughput mtp_t^c (Kbps), VMAF quality qt_t^c (0–100), current playback buffer length bl_t^c (second), segment size ls_t^c (KB), download time dt_t^c (second), percentage of the remaining segments in the video rs_t^c (%), vector of m available sizes for the next segment $\overrightarrow{LS_t^c}$ (KB) and vector of m available VMAF qualities for the next segment $\overrightarrow{QT_t^c}$ (0–100). Instead of feeding the A2C NN the exact values of the input state, we normalize them to enable the agent to generalize the strategy better in an unseen network environment [4].

▷ **Action Space.** The action space $\mathcal{A}$ is defined as the available bitrate levels (*i.e.*, n-dimensional vector) for a given video. In each time epoch t, the `Ahaggar` policy $\pi^{c,\star}$ of agent c maps b_t^c to compact discrete action space $\mathcal{A}$ and select $a_t^{c,\star} \in \mathcal{A}$.

▷ **Observation Space.** We expose a subset of `Ahaggar` states as the observations, where the agent c observes $o_t^c = \{mtp_t^c, qt_t^c, bl_t^c, ls_t^c, dt_t^c, rs_t^c, \overrightarrow{LS_t^c}, \overrightarrow{QT_t^c}\}$ for each time epoch t.

▷ **Output.** The `Ahaggar` *actor* model returns 1×n-dimensional vector representing bitrate levels with their associated probabilities. $\pi^{c,\star} : b_t^c \rightarrow a_t^{c,\star}$ maps the state b_t^c to the best action $a_t^{c,\star}$ based on the state-action probabilities, where $a_t^{c,\star}$ with the highest probability is selected under the current state. The `Ahaggar` *critic* model outputs a single scalar indicating the value function $V^{c,\pi}(b_t^c)$ for the current state.

▷ **NN Architecture.** The `Ahaggar` A2C NN architecture consists of two networks: *actor* and *critic*. Each network uses two 1DConv layers and six linear fully-connected (FC) layers to extract the set of features. Each 1DConv layer consists of 3x3 convolution with feature number (=64) and kernel size (=1) to feed the features $\overrightarrow{LS_t^c}$ and $\overrightarrow{QT_t^c}$. Other inputs are fed into FC layers with feature number (=64) and a Rectified Linear Unit (`ReLU()`) activation function. Then, all input layers are concatenated and finally fed into an FC layer with 64 neurons and a slope of 0.5 to down-sample the concatenated features. The actor and critic use the same structure, but with different outputs. For both networks, we use the Softmax activation function (Softmax()) with L2-norm of networks as the last FC layer, resulting in an output range from 0 to 1.

▷ **Reward Function.** At each time epoch t, the reward r_t^c of an agent c is calculated after each action a_t^c is taken to ensure that `Ahaggar` can learn from past experience. To do so, we adopt a well-know state-of-the-art reward function [48], [12], [51], [33], [25] that linearly combines five metrics (2): perceptual quality ($q_t^c(l_t^c)$), rebuffering duration (rd_t^c) and count (rc_t^c), quality oscillations (qo_t^c) and switches (qs_t^c).

$$r_t^c = \omega_1 \times q_t^c(l_t^c) - \omega_2 \times rd_t^c - \omega_3 \times rc_t^c - \omega_4 \times qo_t^c - \omega_5 \times qs_t^c, \quad (2)$$

where $q_t^c(l_t^c)$ maps the selected bitrate to the quality perceived (VMAF) [51], [12], $qo_t^c = |q_t^c(l_t^c) - q_{t-1}^c(l_{t-1}^c)|$, $qs_t^c = qo_t^c/20$, and ω_i are the coefficients of the reward function. Herein, following prior works [48], [25], we set qs_t^c as the difference of 20 in VMAF values of two consecutive segments. This QoE model is developed based on linear regression on two datasets: Comyco [25] and Waterloo SQoE-IV [20], where 70% of the data is used for training and 30% for testing. We followed the same setup to tune the coefficients and our results show that $\omega_1 = 0.077$, $\omega_2 = 1.249$, $\omega_3 = 2.877$, $\omega_4 = 0.049$, and $\omega_5 = 1.436$ achieve the best trade-off between the five QoE metrics. These results are similar to [48].

▷ **Policy Gradient and Training Algorithm.** The essential objective of `Ahaggar` is to improve the policy via *boosting* the probabilities of high-reward samples from the collected trajectories and *declining* the possibilities of failure samples from the bad trajectories. For every time epoch t, each RL agent c of `Ahaggar` selects the action a_t^c that corresponds to the bitrate for the next segment using the improved policy $\pi : \pi_\theta^{c,\star}(b_t^c, a_t^c) \rightarrow [0, 1]$ at state b_t^c, which results in the best accumulated discounted reward that is expressed as:

$$G_t^c = \sum_{\bar{t}=t}^{T_{\pi_\theta^c}} \gamma^{\bar{t}-t} \times r_t^c, \quad a_t^c = \arg\max_a \mathbb{E}\Big[G_t^c(b_t^c, a)\Big], \quad (3)$$

where G_t^c is computed from time t to the end of training, $T_{\pi_\theta^c}$ denotes the batch size for updating the gradient policy π_θ^c, $\gamma \in [0, 1]$ is the discount factor, θ is the policy parameter, and $\pi_\theta^{c,\star}(b_t^c, a_t^c)$ is the probability that action a_t^c is taken in state b_t^c. DPPO allows `Ahaggar` to run multi-agents (or *workers*), where each agent has its own A2C network and data collection. Thus, the gradient calculations are distributed over workers, as shown in Fig. 1. For each episode, an agent c updates its gradient policy such that G_t^c is maximized with respect to the policy parameters θ, as follows:

$$\bigtriangledown \bar{G}_t^c = \frac{1}{\Theta} \sum_{\theta=1}^{\Theta} \sum_{t=1}^{T_{\pi_\theta^c}} A_t^{\pi_\theta^c}(b_t^c, a_t^c) \bigtriangledown \log \pi_\theta^c(a_t^c, s_t^c), \quad (4)$$

where Θ is the total number of episodes, $A^{\pi_\theta^c}(b_t^c, a_t^c)$ is the *advantage* function that represents the difference in the expected cumulative reward after deterministically selecting the action a_t^c in state b_t^c, compared with the expected reward for action drawn from policy π_θ^c. In PPO, the *advantage* function is calculated as function of G_t^c and baseline base_t^c that has an impact on the convergence of G_t^c. Prior work [52] found that $A^{\pi_\theta^c}$ did not generalize well. Hence, in DPPO, we revise the advantage function by using a truncated backpropagation through time with a window of length κ such that $A^{\pi_\theta^c}(b_t^c, a_t^c) = Q^{\pi_\theta^c}(b_t^c, a_t^c) - V^{\pi_\theta^c}(b_t^c)$. $Q^{\pi_\theta^c}$ is calculated by

the actor network, which uses the κ-step Temporal Difference (TD) approach given by: $Q^{\pi_\theta^c}(b_t^c, a_t^c) = \sum_{\kappa=0}^{\kappa=\Theta-1} \gamma^\kappa r_{t+\kappa}^c + \gamma^\Theta V(b_{t+\Theta}^c)$. For each episode, the agent c of the actor network aims to maximize G_t^c through maximizing $A^{\pi_\theta^c}$, where it samples a trajectory of bitrate decisions and uses the empirically computed advantage as an unbiased estimate of $A^{\pi_\theta^c}(b_t^c, a_t^c)$. To alleviate overfitting issues, `Ahaggar` uses dropouts with probability ($p = 0.5$) to add a regularization term to the update of the actor network. This regularization represents the entropy $\mathcal{E} = \mathbb{H}(\pi_\theta^c(.|b_t^c))$ of the probabilities over the bitrate decisions. Therefore, the parameter θ_{π^c} of the actor is updated via a stochastic gradient ascent using (5).

$$\theta_{\pi^c} \leftarrow \theta_{\pi^c} + \alpha \sum_{t=1}^{T_\theta} A_t^{\pi_\theta^c}(b_t^c, a_t^c) \bigtriangledown_\theta \log \pi_\theta^c(a_t^c, b_t^c) + \beta\, \mathcal{E}, \quad (5)$$

where T_θ is the update interval, α is the learning rate and β is the entropy parameter that is set to a large value at the beginning of the training to encourage exploration and decreases over time to emphasize improving rewards.

To calculate the advantage $A(b_t^c, a_t^c)$ for a given experience, we have to estimate the value function $V^{\pi_\theta^c}(b)$. This estimation is performed by the critic network that makes an objective assessment for all the states $\forall b_t^c \in \mathcal{B}$ of an agent c during the training. To do so, the critic network uses the standard TD method to compute the loss function and minimizes its value. The parameter θ_{v^c} of the critic network is updated through a stochastic gradient descent (SGD) algorithm using (6).

$$\theta_{v^c} \leftarrow \theta_{v^c} - \bar{\alpha} \sum_{t=1}^{T_\theta} \bigtriangledown_\theta (r_t^c + \gamma V^{\pi_\theta^c}(b_{t+1}^c; \theta_{v^c}) - V^{\pi_\theta^c}(b_t^c; \theta_{v^c}))^2, \quad (6)$$

where $\bar{\alpha}$ is the learning rate for the critic, $V^{\pi_\theta^c}(b_t^c; \theta_{v^c})$ and $V^{\pi_\theta^c}(b_{t+1}^c, \theta_{v^c})$ are the objective assessments for b_t^c and b_{t+1}^c, respectively, from the critic network.

Finally, we update the policy π_θ periodically every κ-steps using PPO with constrained clipped objective (CCO) and the Adam optimizer. The constraint represents how much the policy is allowed to change, expressed in terms of the Kullback-Leibler (KL) divergence (KL$[\pi_{\theta_{old}}^c | \pi_\theta^c]$). Hence, the CCO is expressed as: $\theta_{\kappa+1} = \arg\max_\theta \mathcal{L}_{\theta_\kappa}^{KLPEN}(\theta)$, where

$$\mathcal{L}_{\theta_\kappa}^{KLPEN}(\theta) = \mathbb{E}\Big[\sum_{t=1}^{T_\theta} ratio_t(\theta) A_t^{\pi_{\theta_\kappa}^c} - \bar{\beta}\mathrm{KL}[\pi_{\theta_{old}}^c | \pi_\theta^c]\Big], \text{and} \quad (7)$$

$\mathbb{E}$ is the empirical expectation over time steps, $ratio_t(\theta)$ (= $\pi_\theta^c(b_t^c, a_t^c)$/ $\pi_{\theta_{old}}^c(b_t^c, a_t^c)$) is the ratio of the probabilities under the new and old policies, ε is the clip hyperparameter (usually fixed to 0.1) and $\bar{\beta}$ is the KL penalty hyperparameter.

▷ **Multi-agent Training with DPPO.** In the training, `Ahaggar` spawns MARL agents in parallel (Fig. 1). Each agent is configured to run independently with a shared environment such that it experiences a different set of input states from the environment. Here, the N agents continually send their parameters θ to a central agent (termed the chief), which aggregates them to generate a single `Ahaggar` model. For each sequence of parameters θ that it receives, the chief uses the A2C algorithm to compute a gradient based on (5) and (6). Then, the chief updates the A2C networks and pushes out the new model to the agent that sent the parameters. Such update process can happen synchronously or asynchronously among all agents, but we found that averaging gradients and applying them synchronously leads to better results in the meta-testing phase. Our algorithms for the agents and chief are given in [1].

▷ **Meta-Learned Policies for Training Algorithm.** We adopt the MAML approach, which allows learning model parameters θ via meta-RL, *i.e.*, finding the model parameters *sensitive* to changes in the environment, allowing the `Ahaggar` model to achieve fast adaptation to unseen environments during the inference phase. The training algorithm consists of two loops:

(1) Inner Loop. For each episode, each agent c first picks randomly a specific network and content trace as the environment, and sample $X \in \mathcal{D}$ trajectories (also referred to as shots) where $\mathcal{D} = \{(b_1^c, a_1^c); \ldots; (b_k^c, a_k^c)\}$ denotes the set of sampled trajectories for inner loop in that environment according to the current policy π_θ^c. The `Ahaggar` meta-model then is optimized by the collected trajectories with the DPPO and Adam optimizer. In particular, we want to learn θ after a small number κ of policy gradient updates on the data from an environment $Evt_i \sim p(Evts)$ to obtain θ_κ^i. Here, i denotes the index of a particular environment in batch of environments $Evts$. This set of κ updates is called *inner-loop update*. The updated θ_κ^i after κ-step on data from Evt_i is given in (8).

$$\theta_\kappa^i = \theta - \alpha \bigtriangledown_\theta \mathcal{L}_{Evt_i}^{DPPO}(f_\theta, \mathcal{D}), \quad (8)$$

where f_θ is the `Ahaggar` meta-model and $\mathcal{L}_{Evt_i}^{DPPO}(f_\theta)$ is the loss on the environment Evt_i after κ-step of updates.

(2) Outer Loop. For each episode, each agent c continually samples many trajectories ($\in \mathcal{D}_\kappa^i$; the set of sampled trajectories for outer loop) from the randomized environments via meta-policy $\pi_{\theta_\kappa^i}^c$ of meta-model $f_{\theta_\kappa^i}$ and calculates gradients for θ with the trajectory. After that, these agents send the calculated gradients to the chief, which in turn merges them via agents' loss functions and the outer loop's learning rate β. Formally, we define a *meta-objective* ($\mathcal{L}_{meta}(\theta)$) as $\sum_{t=1}^{T_{\pi_\theta^c}} \mathcal{L}_{Evt_i}^{SGD}(f_{\theta_\kappa^i})$. The optimization of $\mathcal{L}$ is called the *outer-loop update*. The resulting update for θ is given by (9).

$$\theta = \theta - \beta \bigtriangledown_\theta \sum_{t=1}^{T_{\pi_\theta^c}} \mathcal{L}_{Evt_i}^{SGD}(f_{\theta_\kappa^i}, \mathcal{D}_\kappa^i), \quad (9)$$

where the update is performed using SGD, β is a learning rate and $\mathcal{L}_{Evt_i}^{SGD}$ denotes the loss on the environment Evt_i.

C. `Ahaggar` Meta-Testing (Online)

The objective of `Ahaggar` is to learn how to adapt to heterogeneous network environments during the online phase through continual learning enabled by MAML. During the meta-testing phase, we use our HAS-based streaming system (Fig. 2), which consists of CMCD/SD-aware DASH clients running in Docker instances and a CMCD/SD-aware Node.js server with an HTTP server and an NJS application. NJS is written in JavaScript and extends the Node.js configuration syntax to implement `Ahaggar`'s bitrate guidance functions and the communication with the dash.js clients. At runtime, for each client, the `Ahaggar` meta-model uses a JSON file that stores the model meta-parameters (θ and θ_κ^i) and trajectories ($\mathcal{D}$ and $\mathcal{D}_\kappa^i$) learned and captured every 40 shots during the offline phase.

IV. Performance Evaluation

A. Ahaggar Implementation

1) Choice of Ahaggar Parameters: To train the Ahaggar model, we used a total of 2000 traces (1500 network and 500 content) from different datasets as described in Section III-B. We randomized them and then used 80% for training and 20% for testing. With an 80–20 train-test split, we performed a 5-fold walk-forward cross-validation on each dataset. Training parameters can impact the performance of Ahaggar, so we empirically set the parameters as summarized in Table II.

2) Offline Training: To train the Ahaggar meta-model, we used a customized trace-based segment-level Gym simulator that is based on Park [32]. This simulator was implemented in Python 3.6 to simulate a typical HAS system based on real-world network and content traces. We used TFLearn 1.5.0 [46], RLlib of Ray 1.12.0 [30] and TensorFlow 2.4.0 to implement Ahaggar's NN architecture and build the training workflow.

3) Online Testing: To test Ahaggar, we implemented a CMCD/SD-enabled streaming system [3] with Ahaggar's bitrate guidance functions. We (i) added new CMCD parameters ($qt, dt, rs, ls, \overrightarrow{QT}, \overrightarrow{LS}$) to support Ahaggar design, and (ii) used the $mb = l$ (maximum suggested bitrate) CMSD-Dynamic parameter to convey Ahaggar's bitrate guidance to each corresponding client. On the server side, we used TensorFlow.js converter [43] to convert and load a pre-trained meta-model into a JavaScript Web-based application and run inference through TensorFlow.js. On the client side, we implemented a simple heuristic as our ABR scheme, which used Ahaggar bitrate guidance decisions to perform rate adaptation. To simplify input state data collection, we appended the manifest files by adding four tags: *size*, *phone*, *hdtv* and *uhdtv*. These tags represent the segment sizes and VMAF scores for phone, HDTV and UHDTV, respectively. The VMAF scores were computed using different VMAF models depending on the device resolution. We provide a sample manifest file in [3].

TABLE II: Ahaggar training/testing parameters.

	Parameter	Symbol	Value
A2C	Discount factor	γ	0.99
	Advantage estimator	λ	1.0
	Actor & Critic learning rates	α, $\bar{\alpha}$	10^{-4}, 10^{-3}
DPPO	Clip parameter & Clip learning rate	ϵ, lr_{clip}	0.2, 0.01
	KL penalty	kl_{target}	0.01
	PPO policy update interval & steps	T_θ, κ-step	k, 20
	Number of workers	N	1000
	Adam learning rate & Adam β	lr, Adam β	3×10^{-5}, 0.999
MAML	Inner-loop learning rate	α	10^{-4}
	Outer-loop learning rate	β	10^{-3}
	Trajectories sample (shots)	X	40
NN	Time step	t	Segment duration
	Batch total time steps in a video	k	Video duration
	Batch size	$T_{\pi_\theta^c}$	k per episode
	Number of episodes & iterations	Θ & itr	3000 & 2000 per episode

B. Methodology and Evaluation Setup

1) Video Sample and Parameters: The HTTP server hosted the 4K DASH dataset [39] that was not used in training. We encoded the 636 seconds long *Big Buck Bunny* (BBB) into four-second segments in FFmpeg using the H.264 codec at 30 fps and in 13 bitrates/resolutions. Further characteristics of BBB are given in [1].

2) Network Traces: We used network traces with different user mobility (bus, walking, car, train, bicycle, tram, ferry and driving) to throttle the bandwidth between the server and clients. These traces were extracted from the 20% of network datasets for testing (Belgium 4G/LTE [49], NYU LTE [34], Lumous 4G/5G [37]). From each dataset, we randomly extracted six network traces where the inter-variation duration between the bandwidth values was fixed to five seconds. The traces are further characterized in [1].

3) ABR Schemes: We compared Ahaggar against heuristics such as throughput-based (TH), buffer-based (BOLA) and Dynamic (TH+BOLA) from dash.js [19] and RobustMPC [51] and one learning-based scheme: Pensieve [33]. The heuristic-based schemes were tuned and Pensieve was retrained with our datasets and QoE metrics to fit each experiment.

4) Performance Metrics: We tested the ABR schemes using two main QoE models: Linear QoE [48] and ITU P.1203 QoE (Mode 0) [42]. For every session, we computed the accumulated QoE^{lin} using a linear function as follows:

$$\omega_1 \sum_{t=1}^{k} q_t^c(l_t^c) - \omega_2 \sum_{t=1}^{k} rd_t^c - \omega_3 rc_t^c - \omega_4 \sum_{t=2}^{k} qo_t^c - \omega_5 \sum_{t=2}^{k} qs_t^c, \quad (10)$$

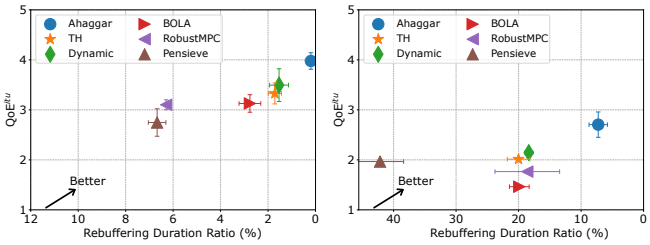
where $\sum_{t=1}^{k} q_t^c(l_t^c)$ is the accumulative perceived perceptual quality, $\sum_{t=1}^{k} rd_t^c$ is the total rebuffering duration (RD), rc_t^c is the total rebuffering count (RC), $\sum_{t=2}^{k} qo_t^c$ is the cumulative quality oscillations, $\sum_{t=2}^{k} qs_t^c$ is the total number of quality switches, and k is the total number of segments. The coefficients of $\omega_{1,2,3,4,5}$ are given in (2). To simplify the presentation of the QoE, we used a normalized QoE^{lin} (N-QoE^{lin}) with values between 0 and 1. To achieve that, we used the best achievable QoE (QoE*) in each session such that N-QoE^{lin} = QoE^{lin} / QoE*. The ITU P.1203 QoE model in Mode 0 (O.46) takes as input four metrics: bitrate, rebuffering duration, frame rate and content resolution. How to compute the QoE^{itu} is described in [42]. This model outputs QoE values in the range of one to five (MOS) and we normalized them (N-QoE^{itu}) to [0,1]. In addition, we computed (i) the total downloaded (TD) size (in MB) metric to measure how much bandwidth was consumed during the session, (ii) percentage of the HD (pHD) segments rendered at 720p or higher, and (iii) percentage of the UHD (pUHD) segments rendered at 2160p.

5) Evaluation Setup and Scenarios: Our setup consisted of one physical machine running Ubuntu 18.04.6 LTS, AMD Ryzen 7 3700X 8-Core CPU and 32 GB memory. We ran a Docker container for each client, in which we ran a CMCD/SD-aware dash.js (v4.2.1) client on a Google Chrome browser (v103) with headless mode enabled using Puppeteer (https://pptr.dev/). The maximum playback buffer level was kept at the default value of 20 seconds. For network emulation, we used tc NetEm (https://man7.org/linux/man-pages/man8/tc-netem.8.html) on the server to throttle the total bandwidth available to the clients according to the network traces described in Section IV-B2; we used a specific client-trace map where for each session, a specific network trace was called. We evaluated Ahaggar in the multi-client scenario with six clients (more details are given in [1]).

C. Results for Multiple Identical Clients

For each ABR scheme, we ran multiple UHDTV clients. Table III shows the total QoE and detailed breakdown of each QoE metric for each ABR scheme for various network traces. We provide the average and standard deviation values for six clients and over five runs in the format of *average*±*std*. In general, `Ahaggar` gained the best possible performance in terms of RC, RD and TD without sacrificing the VMAF score compared to other baselines in all network traces. Looking at the averages across all the network traces, we see that `Ahaggar` reduced average RD by 62.81% (84.36%), average RC by 53.52% (71.18%) and average TD by 53.27% (59.34%) compared to the heuristic-based (learning-based) ABR schemes. In addition, `Ahaggar` significantly reduced the number of times an UHD segment was picked when there was no noticeable VMAF score difference compared to the other best performing schemes (RobustMPC and Dynamic) across all network traces. Such reduction obviously translates to big bandwidth savings (see the Avg. TD column in Table III).

We anticipated these results because `Ahaggar` makes bitrate guidance decisions based on not only the throughput, buffer level and segment sizes, but also segment quality and device resolution. It also uses MAML for continual learning and fast adaptation to unseen environments. In contrast, other ABR schemes use one or more heuristics or an NN combining these heuristics and they do not necessarily perform well in unseen environments. Fig. 3 and Table III confirm this. For instance, Pensieve achieved the highest average selected bitrate and average pUHD, but it performed poorly in most other metrics. In the same context, BOLA failed to deliver a good video quality with inferior VMAF scores and RobustMPC suffered from frequent and long rebuffering events.



(a) Lumous 4G. (b) Belgium 4G/LTE.

(c) NYU LTE. (d) Lumous 5G.

Fig. 3: Avg. QoE^{itu} and avg. rebuffering duration ratio in various network traces. The bottom (left) edge, mark and top (right) edge indicate the mean+std, mean and mean−std, respectively, with a 95% confidence interval (CI).

Similarly, `Ahaggar` achieved the highest average QoE^{itu} and lowest average rebuffering duration (see Fig. 3). In detail, `Ahaggar` achieved the highest average QoE with an improvement of [Lumous 4G 3a: 22.28% (44.73%), Belgium 4G/LTE 3b: 49.49% (37.06%), NYU LTE 3c: 55.04% (85.08%), Lumous 5G 3d: 8.01% (31.10%)] and lowest average rebuffering duration with a reduction of 62.81% (84.36%) across all network traces, compared to heuristic-based (learning-based) ABR schemes. Compared to `Ahaggar`, Dynamic achieved the second best average results in terms of the QoE and rebuffering duration. This is because of the Dynamic design that combines the benefits of BOLA and TH by switching between both of them in runtime based on the stability of the current buffer level. However, Pensieve followed by RobustMPC suffered from low QoE and long RD due to wrong ABR decisions. It is worth mentioning that all schemes faced few rebuffering events in Lumous 5G because sometimes the bandwidth dropped significantly and suddenly caused by the handoffs to 4G. This is a behaviour known in 5G networks operating in higher frequencies [37].

To understand how QoE^{itu} (Mode 0) is computed for each session, Table III (the eighth and ninth columns) highlights the scores of its essential metrics (O.23: Rebuffing Duration Score and O.46: Overall Score) for different ABR schemes. The score of each metric is given in the MOS range of one to five. Here, we deduce three important thrusts. First, `Ahaggar` outperformed the baselines achieving the best O.23 and O.46 scores for all network traces with an average improvement of 67.55% (heuristic-based: 60.75%, learning-based: 94.75%) and 36.86% (heuristic-based: 33.70%, learning-based: 49.49%) across all network traces, respectively. It also achieved higher O.35 (Visual Quality Score, not shown) scores with values ranging between 4.60 and 4.94. These results confirm how well `Ahaggar` performs to balance the QoE^{itu} metrics. Second, the Belgium 4G/LTE dataset has the lowest bandwidth values in its network traces. Therefore, all ABR schemes achieved the lowest scores in terms of O.23, O.35 and O.46. Nonetheless, since `Ahaggar` has been designed to adapt quickly to challenging network conditions (thanks to MAML), it was able to obtain the best O.23 (2.37) and O.46 (2.70) scores. Although other baselines achieved a comparable O.35 score (not shown), they faced frequent and long rebuffering events due to their greedy bitrate selection strategy. Third, Dynamic was the runner-up, receiving the second best results in terms of O.23 and O.46. Unexpectedly, Pensieve failed to produce good ABR decisions, leading to multiple rebuffering events that contributed to the lowest O.23 score, which impacted O.46 negatively in most network traces.

We also conducted a comparison between QoE^{itu} and QoE^{lin}. We first normalized both values (Section IV-B4) and the comparison between different ABR schemes for various network traces is listed in last column of Table III. In each network trace, `Ahaggar` achieved the highest and consistent performance in terms of N-QoE^{itu} and N-QoE^{lin} (only in NYU LTE, TH and Dynamic were slightly better) compared to other ABR schemes. We can see that the N-QoE^{itu} and N-QoE^{lin} are almost identical for each dataset, and thus, can be used interchangeably in practice.

TABLE III: Average results of the QoE and its metrics for different network traces. ↑: higher is better (green), ↓: lower is better (green), lowest performance (red), format: *average*±*std*.

		Avg. Selected Bitrate (Mbps)	Avg. Perceptual Quality (VMAF)	Average RD (s)	Average RC (#)	Average TD (MB)	Average pHD (%)	Average pUHD (%)	Avg. QoE^{itu} (O.23)	Avg. QoE^{itu} (O.46)	N-QoE^{itu} (N-QoE^{lin})
Lumous 4G	Ahaggar	08.17±0.85	88.94±0.74	**01.25±0.86** ↓	**03.30±1.24** ↓	**762.18±72.24** ↓	**49.74±7.59** ↑	46.65±7.62	**3.82±0.36** ↑	**3.98±0.27** ↑	**1.00** ↑ (**1.00** ↑)
	TH	28.81±3.20	97.67±0.67	10.90±8.43	07.30±3.74	2426.03±220.56	2.59±3.66	95.07±4.01	2.94±0.66	3.33±0.49	0.84 (0.92)
	BOLA	29.96±2.52	97.71±0.76	17.56±15.93	08.70±4.55	2448.60±172.58	0.79±1.47	95.62±2.91	2.73±0.78	3.13±0.59	0.79 (0.89)
	Dynamic	29.18±3.19	97.73±0.59	09.78±9.01	06.33±4.06	2417.28±191.59	2.19±3.15	95.28±3.71	3.16±0.68	3.49±0.51	0.88 (0.91)
	RobustMPC	**31.96±0.007** ↑	**98.23±0.00** ↑	40.02±47.27	12.40±11.83	2536.18±00.00	0.00±0.00	98.41±0.29	2.67±1.14	3.10±0.86	0.78 (0.84)
	Pensieve	31.95±0.004	98.08±0.07	42.45±50.52	16.47±12.14	2503.77±01.11	0.02±0.04	**98.43±0.36** ↑	2.19±0.85	2.75±0.65	0.69 (0.81)
Belgi. 4G/LTE	Ahaggar	2.63±0.29	74.67±5.48	**46.15±22.09** ↓	**10.90±4.38** ↓	**242.46±20.51** ↓	87.31±8.27	0.93±0.57	**2.37±0.62** ↑	**2.70±0.47** ↑	**1.00** ↑ (**1.00** ↑)
	TH	3.28±0.16	78.39±2.27	127.24±50.00	25.67±6.94	304.19±14.99	88.42±4.72	0.92±0.14	1.33±0.30	2.01±0.21	0.74 (0.94)
	BOLA	1.74±0.45	38.38±4.78	126.48±37.15	53.53±13.91	310.22±26.46	34.32±6.56	0.33±0.15	1.03±0.04	1.46±0.10	0.54 (0.44)
	Dynamic	3.31±0.14	**78.90±2.51** ↑	116.83±56.14	22.27±7.85	300.45±14.59	**89.22±5.41** ↑	0.94±0.15	1.49±0.42	2.14±0.31	0.79 (0.96)
	RobustMPC	2.65±0.28	66.50±3.66	118.30±50.00	37.80±10.36	317.81±28.91	65.15±6.68	3.05±1.34	1.12±0.10	1.77±0.07	0.65 (0.87)
	Pensieve	**4.08±0.36** ↑	70.30±8.09	268.14±112.47	19.40±7.53	344.13±25.45	72.06±11.47	**4.06±0.86** ↑	1.57±0.40	1.97±0.34	0.73 (0.57)
NYU LTE	Ahaggar	4.13±0.90	83.61±2.52	**18.98±16.57** ↓	**7.00±3.74** ↓	**406.58±92.05** ↓	**81.81±8.15** ↑	14.41±8.39	**3.03±0.72** ↑	**3.35±0.54** ↑	**1.00** ↑ (0.94)
	TH	8.98±2.90	89.93±3.22	35.77±25.34	12.87±5.87	867.87±274.48	44.24±15.61	51.23±17.35	2.15±0.52	2.73±0.41	0.81 (**1.00** ↑)
	BOLA	11.81±2.91	75.95±11.49	113.37±61.35	41.50±19.97	1080.36±258.57	21.33±9.40	53.50±18.61	1.26±0.40	1.79±0.26	0.53 (0.78)
	Dynamic	9.56±2.98	**90.30±3.14** ↑	34.43±21.70	10.90±5.56	895.51±272.35	40.69±14.54	54.62±16.35	2.40±0.59	2.90±0.46	0.87 (**1.00** ↑)
	RobustMPC	16.06±2.60	84.07±8.48	408.16±62.34	98.40±20.58	1404.23±196.54	20.18±14.91	**60.37±16.87** ↑	1.00±0.00	1.72±0.12	0.51 (0.28)
	Pensieve	**18.09±5.35** ↑	88.27±6.98	753.57±288.70	85.07±25.53	1696.57±394.50	36.80±14.78	53.58±16.61	1.00±0.00	1.81±0.06	0.54 (0.02)
Lumous 5G	Ahaggar	11.62±1.16	94.36±2.12	**5.49±7.39** ↓	**3.60±1.93** ↓	**952.02±68.60** ↓	**7.14±14.94** ↑	89.19±15.07	**3.75±0.51** ↑	**3.92±0.38** ↑	**1.00** ↑ (**1.00** ↑)
	TH	31.86±0.18	98.17±0.13	8.76±8.21	6.70±1.88	2535.57±10.17	0.25±0.57	97.97±0.86	3.03±0.35	3.40±0.26	0.87 (0.98)
	BOLA	31.84±0.11	97.47±1.02	7.94±8.72	4.77±1.92	2556.61±19.55	0.79±1.03	95.49±2.85	3.42±0.41	3.66±0.31	0.93 (**1.00** ↑)
	Dynamic	31.90±0.12	98.14±0.17	7.24±8.41	3.63±1.63	2537.78±9.46	0.37±0.74	97.60±1.61	3.68±0.43	3.89±0.33	0.99 (0.99)
	RobustMPC	31.95±0.005	**98.24±0.03** ↑	7.64±8.52	5.13±1.81	2536.17±0.015	0.00±0.00	**98.47±0.27** ↑	3.31±0.41	3.60±0.31	0.92 (0.98)
	Pensieve	**31.96±0.005** ↑	98.10±0.10	13.77±9.68	11.67±3.63	2520.09±0.57	0.08±0.18	98.41±0.43	2.49±0.50	2.99±0.37	0.76 (0.95)
Avg. ALL	Ahaggar	06.64±0.80	85.39±2.71	**17.97±11.73** ↓	**6.20±2.82** ↓	**590.81±63.35** ↓	**56.50±9.74** ↑	37.79±7.91	**3.24±0.55** ↑	**3.49±0.42** ↑	**1.00** ↑ (**0.99** ↑)
	TH	18.23±1.61	91.04±1.57	45.67±22.99	13.13±4.61	1533.41±130.05	33.87±6.14	61.30±5.59	2.36±0.46	2.87±0.34	0.82 (0.96)
	BOLA	18.84±1.50	77.38±4.51	66.34±30.79	27.12±10.09	1598.95±119.29	14.31±4.61	61.23±6.13	2.11±0.54	2.51±0.32	0.70 (0.78)
	Dynamic	18.49±1.61	**91.27±1.60** ↑	42.07±23.82	10.78±4.77	1537.75±122.00	33.12±5.96	62.11±5.45	2.68±0.53	3.11±0.40	0.88 (0.96)
	RobustMPC	20.65±0.72	86.76±3.04	143.53±42.03	38.43±11.14	1698.60±56.37	21.33±5.40	**65.07±4.69** ↑	2.03±0.41	2.55±0.34	0.72 (0.74)
	Pensieve	**21.52±1.43** ↑	88.69±3.81	269.48±115.34	33.15±12.21	1766.14±105.41	27.24±6.62	63.62±4.56	1.81±0.44	2.38±0.36	0.68 (0.59)

D. Results for Multiple Diverse Clients

To evaluate the effectiveness of Ahaggar in adapting to different device resolutions (DR), we ran two clients with each DR (total of six). Table IV highlights the results over five runs. The key takeaway is that Ahaggar achieved different average results for each DR, confirming Ahaggar's DR awareness. Ahaggar picked a higher bitrate on the average for a UHDTV compared to an HDTV and a phone. For instance, it selected 1.5x-2x higher bitrate for UHDTV compared to the phone with almost a 1-JND difference between the VMAF scores for various network traces. This is because devices with a phone-like resolution can achieve the highest VMAF score (95-98) requiring only half of the bitrate that a UHDTV requires. We note that the VMAF score differences at a similar bitrate level (*e.g.*, phone vs. HDTV in NYU LTE) are due to the different per-device VMAF models used to calculate the scores.

TABLE IV: Average QoE^{itu} (O.46) scores and its metrics produced by Ahaggar running on devices with different resolutions. ↑: higher is better (green), ↓: lower is better (green), format: *average*±*std*.

	Avg. Selected Bitrate (Mbps)	Avg. Perceptual Quality (VMAF)	Avg. RD (s)	Avg. TD (MB)	Avg. QoE^{itu} (O.46)
Lumous 4G					
Phone	03.61±0.19	**95.20±0.23** ↑	**0.61±0.52** ↓	**316.91±11.85** ↓	4.30±0.24
HDTV	03.81±0.21	88.82±0.30	0.86±1.20	319.65±16.68	**4.35±0.30** ↑
UHDTV	**08.17±0.85** ↑	88.94±0.74	1.25±0.86	762.18±72.24	3.98±0.27
Belgium 4G/LTE					
Phone	**2.83±0.34** ↑	**90.72±3.07** ↑	49.62±35.42	244.74±15.46	2.82±0.59
HDTV	2.61±0.34	82.24±4.45	**42.25±23.02** ↓	**240.20±18.78** ↓	**2.90±0.57** ↑
UHDTV	2.63±0.29	74.67±5.48	46.15±22.09	242.46±20.51	2.70±0.47
NYU LTE					
Phone	03.26±0.21	**94.58±0.79** ↑	**5.24±5.54** ↓	**269.32±11.58** ↓	**3.99±0.64** ↑
HDTV	03.33±0.13	87.99±0.57	5.78±7.06	270.26±12.05	3.88±0.58
UHDTV	**04.13±0.90** ↑	83.61±2.52	18.98±16.57	406.58±92.05	3.35±0.54
Lumous 5G					
Phone	07.62±2.81	**97.26±1.31** ↑	5.40±7.29	**627.18±213.14** ↓	**3.97±0.45** ↑
HDTV	09.78±2.66	94.95±3.00	**5.33±7.11** ↓	797.95±199.95	3.90±0.26
UHDTV	**11.62±1.16** ↑	94.36±2.12	5.49±7.39	952.02±68.60	3.92±0.38
Avg. ALL					
Phone	4.20±0.89	**94.44±1.35** ↑	15.22±12.19	**364.54±63.00** ↓	**3.77±0.48** ↑
HDTV	4.88±0.84	88.5±2.08	**13.55±9.60** ↓	407.01±61.61	3.76±0.43
UHDTV	**6.64±0.8** ↑	85.40±2.72	17.97±11.73	590.81±63.35	3.49±0.42

V. Conclusions

This paper presented Ahaggar, a server-side, learning-based, quality-aware bitrate guidance solution that complements the client-side heuristic-based ABR schemes. Ahaggar adopts two key enablers: (i) a meta-RL approach to find the best bitrate for each client under the given circumstances and quickly adapt to changing network conditions, and (ii) CMCD/SD specification to simplify the metadata exchange between the server and clients. Experiments show that Ahaggar delivers better user experience with less bandwidth consumption over a variety set of network conditions.

Acknowledgements– This research is supported by Singapore MoE Academic Research Fund Tier 2 under MOE's official grant number T2EP20221-0023 and by the Scientific and Technological Research Council of Turkiye under grant number 120C154.

References

[1] Ahaggar Appendix. [Online] Available: https://github.com/NUStreaming/Ahaggar/blob/master/Appendix_Ahaggar.pdf. Accessed on Dec. 22, 2022.

[2] ISO/IEC 23009-5:2017 Information technology — Dynamic adaptive streaming over HTTP (DASH) — Part 5: Server and network assisted DASH (SAND). [Online] Available: https://www.iso.org/standard/69079.html. Accessed on Dec. 22, 2022.

[3] A. Bentaleb, M. Lim, M. N. Akcay, Ali C. Begen, and R. Zimmermann. Ahaggar Bitrate Guidance. [Online] Available: https://github.com/NUStreaming/Ahaggar. Accessed on Dec. 22, 2022.

[4] S. Abbasloo, C.-Y. Yen, and H. J. Chao. Classic Meets Modern: A Pragmatic Learning-based congestion Control for the Internet. In *ACM SIGCOMM*, 2020.

[5] Akamai Inc. EnvivioDash3. [Online] Available: https://dash.akamaized.net/envivio/EnvivioDash3. Accessed on Dec. 22, 2022.

[6] S. Akhshabi, L. Anantakrishnan, C. Dovrolis, and A. C. Begen. Server-based Traffic Shaping for Stabilizing Oscillating Adaptive Streaming Players. In *ACM NOSSDAV*, 2013 (DOI: 10.1145/2460782.2460786).

[7] E. Alpaydin. *Introduction to machine learning*. MIT press, 2020.

[8] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath. Deep Reinforcement Learning: A Brief Survey. *IEEE Signal Processing Mag.*, 34(6):26–38, 2017.

[9] A. C. Begen, M. N. Akcay, A. Bentaleb, and A. Giladi. Adaptive streaming of content-aware-encoded videos in dash. js. *SMPTE Motion Imaging Jour.*, 131(4):30–38, 2022 (DOI: 10.5594/JMI.2022.3160560).

[10] A. C. Begen, A. Bentaleb, D. Silhavy, S. Pham, R. Zimmermann, and W. Law. Road to salvation: streaming clients and content delivery networks working together. *IEEE Communications Mag.*, 59(11):123–128, 2021 (DOI: 10.1109/MCOM.121.2100137).

[11] A. Bentaleb, A. C. Begen, S. Harous, and R. Zimmermann. Data-driven Bandwidth Prediction Models and Automated Model Selection for Low Latency. *IEEE Trans. Multimedia*, 23:2588–2601, 2021 (DOI: 10.1109/TMM.2020.3013387).

[12] A. Bentaleb, A. C. Begen, and R. Zimmermann. SDNDASH: Improving QoE of HTTP Adaptive Streaming using Software Defined Networking. In *ACM Multimedia*, 2016 (DOI: 10.1145/2964284.2964332).

[13] A. Bentaleb, M. Lim, M. N. Akcay, A. C. Begen, and R. Zimmermann. Common Media Client Data (CMCD): Initial Findings. In *ACM NOSSDAV*, 2021 (DOI: 10.1145/3458306.3461444).

[14] 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*, 21(1):562–585, 2019 (DOI: 10.1109/COMST.2018.2862938).

[15] A. Bentaleb, P. K. Yadav, W. T. Ooi, and R. Zimmermann. Dq-dash: A queuing theory approach to distributed adaptive video streaming. *ACM TOMM*, 16(1):1–24, 2020.

[16] A. Bokani, M. Hassan, S. Kanhere, and X. Zhu. Optimizing HTTP-based Adaptive Streaming in Vehicular Environment Using Markov Decision Process. *IEEE Trans. Multimedia*, 17(12):2297–2309, 2015.

[17] Consumer Technology Association. Web Application Video Ecosystem – Common Media Client Data (CMCD), 2020.

[18] Consumer Technology Association. Web Application Video Ecosystem – Common Media Server Data (CMSD), 2022.

[19] DASH-IF. DASH Reference Client. [Online] Available: https://reference.dashif.org/dash.js/. Accessed on Dec. 22, 2022.

[20] Z. Duanmu, W. Liu, Z. Li, D. Chen, Z. Wang, Y. Wang, and W. Gao. Assessing the Quality-of-Experience of Adaptive Bitrate Video Streaming. *arXiv:2008.08804*, 2020.

[21] C. Finn, P. Abbeel, and S. Levine. Model-agnostic Meta-learning for Fast Adaptation of Deep Networks. In *PMLR ICML*, 2017.

[22] A. Ganjam, J. Jiang, X. Liu, V. Sekar, F. Siddiqi, I. Stoica, J. Zhan, and H. Zhang. C3: Internet-scale control plane for video quality optimization. In *USENIX NSDI*, 2015.

[23] Google Brain. TensorFlow. [Online] Available: https://www.tensorflow.org/. Accessed on Dec. 22, 2022.

[24] N. Heess, D. TB, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, S. Eslami, et al. Emergence of Locomotion Behaviours in Rich Environments. *arXiv:1707.02286*, 2017.

[25] T. Huang, C. Zhou, R.-X. Zhang, C. Wu, X. Yao, and L. Sun. Comyco: Quality-aware Adaptive Video Streaming via Imitation Learning. In *ACM Multimedia*, 2019.

[26] J. Jiang, V. Sekar, H. Milner, D. Shepherd, I. Stoica, and H. Zhang. CFA: A Practical Prediction System for Video QoE Optimization. In *USENIX NSDI*, 2016.

[27] J. Kim, Y. Jung, H. Yeo, J. Ye, and D. Han. Neural-enhanced Live Streaming: Improving Live Video Ingest via Online Learning. In *ACM SIGCOMM*, 2020.

[28] Z. Li, A. C. Begen, J. Gahm, Y. Shan, B. Osler, and D. Oran. Streaming video over HTTP with consistent quality. In *ACM MMSys*, 2014 (DOI: 10.1145/2557642.2557658).

[29] 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 Jour. Selected Areas in Communications*, 32(4):719–733, 2014 (DOI: 10.1109/JSAC.2014.140405).

[30] E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, K. Goldberg, J. Gonzalez, M. Jordan, and I. Stoica. RLlib: Abstractions for Distributed Reinforcement Learning. In *PMLR ICML*, 2018.

[31] M. Lim, M. N. Akcay, A. Bentaleb, A. C. Begen, and R. Zimmermann. The Benefits of Server Hinting when DASHing or HLSing. In *ACM MHV*, 2022 (DOI: 10.1145/3510450.3517317).

[32] H. Mao, P. Negi, A. Narayan, H. Wang, J. Yang, H. Wang, R. Marcus, M. Khani Shirkoohi, S. He, V. Nathan, et al. Park: An Open Platform for Learning-augmented Computer Systems. *Advances in Neural Information Processing Systems*, 32, 2019.

[33] H. Mao, R. Netravali, and M. Alizadeh. Neural Adaptive Video Streaming with Pensieve. In *ACM SIGCOMM*, 2017.

[34] L. Mei, R. Hu, H. Cao, Y. Liu, Z. Han, F. Li, and J. Li. Realtime Mobile Bandwidth Prediction Using LSTM NN. In *Springer PAM*, 2019.

[35] V. V. Menon, H. Amirpour, M. Ghanbari, and C. Timmerer. Opte: Online per-title encoding for live video streaming. In *IEEE ICASSP*, 2022.

[36] M. Mu, M. Broadbent, A. Farshad, N. Hart, D. Hutchison, Q. Ni, and N. Race. A Scalable User Fairness Model for Adaptive Video Streaming over SDN-assisted Future Networks. *IEEE Jour. Selected Areas in Communications*, 34(8):2168–2184, 2016.

[37] A. Narayanan, X. Zhang, R. Zhu, A. Hassan, S. Jin, X. Zhu, X. Zhang, D. Rybkin, Z. Yang, , et al. A Variegated Look at 5G in the Wild: Performance, Power, and QoE Implications. In *ACM SIGCOMM*, 2021.

[38] S. Pham, P. Heeren, D. Silhavy, and S. Arbanowski. Evaluation of Shared Resource Allocation Using SAND for ABR Streaming. In *ACM MMSys*, 2019.

[39] J. J. Quinlan and C. J. Sreenan. Multi-profile Ultra High Definition (UHD) AVC and HEVC 4K DASH Datasets. In *ACM MMSys*, 2018.

[40] R. Rassool. VMAF Reproducibility: Validating a Perceptual Practical Video Quality Metric. In *IEEE BMSB*, 2017.

[41] H. Riiser, P. Vigmostad, C. Griwodz, and P. Halvorsen. Commute Path Bandwidth Traces from 3G Networks: Analysis and Applications. In *ACM MMSys*, 2013.

[42] W. Robitza, S. Göring, A. Raake, D. Lindegren, G. Heikkilä, J. Gustafsson, P. List, B. Feiten, U. Wüstenhagen, M.-N. Garcia, et al. HTTP Adaptive Streaming QoE Estimation with ITU-T Rec. P. 1203: Open Databases and Software. In *ACM MMSys*, 2018.

[43] D. Smilkov, N. Thorat, Y. Assogba, C. Nicholson, N. Kreeger, P. Yu, S. Cai, E. Nielsen, D. Soegel, S. Bileschi, et al. Tensorflow.js: Machine Learning for the Web and Beyond. *Proceedings of Machine Learning and Systems*, 1:309–321, 2019.

[44] K. Spiteri, R. Sitaraman, and D. Sparacio. From Theory to Practice: Improving Bitrate Adaptation in the DASH Reference Player. *ACM TOMM*, 15(2s):1–29, 2019.

[45] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman. BOLA: Near-optimal Bitrate Adaptation for Online Videos. *IEEE/ACM Trans. Networking*, 28(4):1698–1711, 2020.

[46] Y. Tang. TFLearn: TensorFlow's High-level Module for Distributed Machine Learning. *arXiv:1612.04251*, 2016.

[47] F. Tashtarian, A. Bentaleb, A. Erfanian, H. Hellwagner, C. Timmerer, and R. Zimmermann. HxL3: Optimized Delivery Architecture for HTTP Low-Latency Live Streaming. *IEEE Trans. Multimedia*, 2022.

[48] B. Turkkan, T. Dai, A. Raman, T. Kosar, C. Chen, M. Bulut, J. Zola, and D. Sow. GreenABR: Energy-Aware Adaptive Bitrate Streaming with Deep Reinforcement Learning. In *ACM MMSys*, 2022.

[49] J. van der Hooft, S. Petrangeli, T. Wauters, R. Huysegems, P. R. Alface, T. Bostoen, and F. De Turck. HTTP/2-Based Adaptive Streaming of HEVC Video Over 4G/LTE Networks. *IEEE Communications Letters*, 20(11):2177–2180, Nov. 2016.

[50] 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 *USENIX NSDI*, 2020.

[51] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli. A Control-theoretic Approach for Dynamic Adaptive Video Streaming over HTTP. In *ACM SIGCOMM*, 2015.

[52] C. Yu, A. Velu, E. Vinitsky, Y. Wang, A. Bayen, and Y. Wu. The Surprising Effectiveness of PPO in Cooperative, Multi-Agent Games. *arXiv:2103.01955*, 2021.

[53] 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 *ACM MobiCom*, 2020.

[54] C. Zhu, M. Dastani, and S. Wang. A Survey of Multi-Agent Reinforcement Learning with Communication. *arXiv:2203.08975*, 2022.

[55] X. Zhu, S. Sen, and Z. M. Mao. Livelyzer: Analyzing the First-Mile Ingest Performance of Live Video Streaming. In *ACM MMSys*, 2021.