PiTree: Practical Implementation of ABR Algorithms Using Decision Trees

Zili Meng[1,4], Jing Chen[1,2,4], Yaning Guo[1,2,4], Chen Sun[3,4], Hongxin Hu[5], Mingwei Xu[1,3,4]
[1]Institute for Network Science and Cyberspace, Tsinghua University
[2]Department of Electronic Engineering, Tsinghua University
[3]Department of Computer Science and Technology, Tsinghua University
[4]Beijing National Research Center for Information Science and Technology (BNRist)
[5]School of Computing, Clemson University
zilim@ieee.org, {j-chen16, gyn17, c-sun14}@mails.tsinghua.edu.cn, hongxih@clemson.edu, xumw@tsinghua.edu.cn

ABSTRACT

Major commercial client-side video players employ adaptive bitrate (ABR) algorithms to improve user quality of experience (QoE). With the evolvement of ABR algorithms, increasingly complex methods such as neural networks have been adopted to pursue better performance. However, these complex methods are too heavyweight to be directly implemented in client devices, especially mobile phones with very limited resources. Existing solutions suffer from a trade-off between algorithm performance and deployment overhead. To make the implementation of sophisticated ABR algorithms practical, we propose PiTree, a *general*, *high-performance* and *scalable* framework that can faithfully convert sophisticated ABR algorithms into lightweight *decision trees* to reduce deployment overhead. We also provide a theoretical upper bound on the optimization loss during the conversion. Evaluation results on three representative ABR algorithms demonstrate that PiTree could faithfully convert ABR algorithms into decision trees with <3% average performance degradation. Moreover, comparing to original implementation solutions, PiTree could save operating expenses for large content providers.

CCS CONCEPTS

• **Information systems** → **Multimedia streaming**; • **Computing methodologies** → *Sequential decision making*.

KEYWORDS

ABR; practicality; client-side implementation; decision tree

ACM Reference Format:
Zili Meng, Jing Chen, Yaning Guo, Chen Sun, Hongxin Hu, Mingwei Xu. 2019. PiTree: Practical Implementation of ABR Algorithms Using Decision Trees. In *Proceedings of the 27th ACM International Conference on Multimedia (MM '19), October 21–25, 2019, Nice, France.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3343031.3350866

MM '19, October 21–25, 2019, Nice, France

ACM ISBN 978-1-4503-6889-6/19/10…$15.00
https://doi.org/10.1145/3343031.3350866

Table 1: Representative ABR Algorithms

Publication Year: 2012			2018
Rate-based	Festive [38]	Panda [44]	Squad [61]
Buffer-based	BBA [34]	BOLA [56]	BOLA-E [55]
Hybrid (Non-ML)	Elastic [25]	RobustMPC [65]	Quetra [63]
Hybrid (ML)	CS2P [58]	Pensieve [47]	HotDASH [53]

1 INTRODUCTION

In recent years, video streaming traffic plays a prominent role in Internet traffic [36]. Meanwhile, online video clients have increasingly higher demands on the video quality of experience (QoE) [47], which directly correlates with content provider revenue [14, 27]. Therefore, as presented in Table 1, a series of adaptive bitrate (ABR) algorithms are proposed to optimize the video quality, some of which have already been used by content providers [55, 56]. These algorithms usually run on client-side video players [47] that dynamically select a bitrate based on network conditions. ABR algorithms have to handle complicated situations including different QoE demands [47, 56, 64], high variation of network throughput [33, 58], and the cascading effect between actions [47]. Therefore, sophisticated algorithms such as Mixed Integer Linear Programming (MILP) [65], Lyapunov optimization [56], and neural networks [32, 45, 47, 53, 64] are adopted to improve ABR performance.

However, the expensive computation overhead of increasingly complex ABR algorithms prevents them from traditional in-player implementations [47, 65]. Especially, a sharply increasing number of users choose to play videos through smart TVs [53] and mobiles devices such as pads or cellphones [49, 62]. The latest statistics indicate that mobile devices account for 62% of online video views in 2018, and this number is increasing rapidly [4]. These mobile devices often have very limited computation resources, which cannot satisfy the resource requirements of solving complex optimization problems [65, §2.2]. Thus, it is difficult to directly integrate ABR algorithms into HTML pages and implement them in client-side video players [11]. The situation becomes even worse when pursuing higher performance with even more complex optimizations in future research.

To address this problem, ABR algorithm designers propose many solutions, which, however, fail to achieve *high performance* and *scalability* at the same time. Two main categories of solutions include [§2.2]: 1) *Compromising performance to reduce overhead.* RobustMPC [65] provides an online version of its MILP-based algorithm (FastMPC) by enumerating some situations and constructing

a solution table for online lookup. FastMPC, however, is a case-specific method for RobustMPC and actually drastically degrades the performance below many strawman baselines [65]. 2) *Offloading computation to remote ABR servers.* Pensieve [47] and HotDASH [53] suggest offloading the heavyweight computation to remote ABR servers, requiring video clients to send requests to the ABR servers when they need to make decisions. The content providers, however, have to introduce and maintain additional servers to specifically provide ABR optimization services. The operating expenses (OPEX) will be significantly increased. Such solutions cannot scale to real-world large-scale deployment.

Inspired by recent advances in *interpreting* complicated models [16, 46, 50, 66], we could implement sophisticated algorithms in practice by *converting* them into other faithful but lightweight representations. Our key observation is that some representations of algorithms (*e.g.*, decision tree) have similar expressiveness but are more lightweight and efficient compared to those sophisticated representations [15, 16].

Based on this observation, we propose PiTree, a *general*, *high-performance*, and *scalable* framework to bridge the gap between offline design and online implementation of ABR algorithms using *decision trees*. The key idea of PiTree is to faithfully convert sophisticated algorithms into lightweight decision trees to simultaneously achieve high performance and implementation scalability. Due to the high dimension of the decision space of ABR algorithms [§3.1] and the cascading effects between actions [47], however, we are challenged to faithfully and efficiently convert sophisticated ABR algorithms into decision trees. In response, PiTree innovatively adopts *imitation learning* [35] to faithfully convert ABR algorithms to decision trees and employs a *virtual player* to efficiently collect unbiased data. The original ABR algorithm acts like a teacher who continuously corrects the actions of the decision tree (student). We also provide a theoretical analysis of the upper bound of the optimization loss when implementing the decision tree to process online videos.

This paper makes the following contributions:

- We illustrate the problem of practically implementing sophisticated ABR algorithms for content providers [§2.2]. We then propose PiTree, a general, high-performance, and scalable framework for practical implementation of ABR algorithms using decision trees [§3].
- We present an imitation learning-based decision tree training algorithm [§3.2] and provide a theoretical analysis on the upper bound the average optimization loss of PiTree [§3.3].
- We evaluate PiTree with both simulations and real-world implementations. Results show that PiTree could convert state-of-the-art ABR algorithms into lightweight decision trees with negligible performance degradation and little overhead, thus save millions of dollars for content providers [§4].

To the best of our knowledge, PiTree is the first general framework to make it practical to implement state-of-the-art sophisticated ABR algorithms on client-side video players. PiTree is available at https://transys.io/pitree. We believe that PiTree will accelerate the deployment of new sophisticated ABR algorithms with higher performance.

2 BACKGROUND AND MOTIVATION

2.1 ABR Streaming

Dynamic Adaptive Streaming over HTTP (DASH) [57] is the predominant streaming video delivery method today. In DASH systems, each video is partitioned into chunks (*e.g.*, 4-second blocks), where each chunk is encoded in multiple bitrates. A higher bitrate indicates higher video quality. When a user plays a video on the client-side player, the ABR algorithm decides the appropriate bitrate to download for the next chunk and downloads the video chunk into the playback buffer on video clients. As shown in Table 1, ABR algorithms can be categorized into *rate-based* methods, which make decisions with network throughput information, *buffer-based* methods, which make decisions based on video player buffer, and *hybrid* methods, utilizing information from network and buffer.

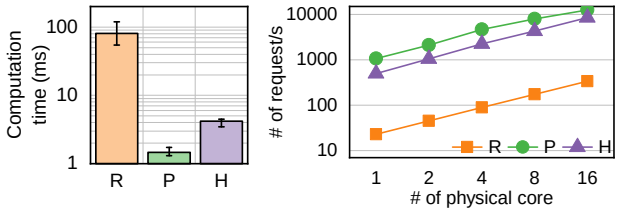
It is well-established that higher QoE follows from 1) higher average video bitrate, 2) fewer rebuffering events, and 3) higher video bitrate smoothness [47, 53, 64]. These factors, however, are often conflicting with each other in the real world. For example, in a network with highly fluctuating throughput, a conservative policy to minimize rebuffering events may lead to lower average bitrate. Meanwhile, the instability of network conditions makes a precise prediction for future bandwidths challenging. Moreover, bitrate selection for a single chunk will affect the future states of video players, which is known as the *cascading effect* of ABR systems [47]. All these factors make the optimization of QoE exceptionally challenging. Existing solutions have already achieved significant improvements in addressing conflicts above. However, since there still exists a gap between current ABR algorithms and offline optimal [14, 47], further evolvement of ABR algorithms are still needed for better performance [24].

2.2 Motivation

With the increase of algorithm complexity, sophisticated ABR algorithms are difficult to be deployed into client-side video players in practice. First, loading heavyweight computation models (*e.g.*, neural networks [32, 45, 47, 53]) will drastically increase the page load time by tens of seconds [§4.3.1], which might make impatient users leave the page [41]. Second, solving complex optimizations on end devices with constrained computation resources will introduce excessive decision latency by up to 1000x [§4.3.2], which might exceed the video chunk lengths (often 2 to 4 seconds [38]) and severely degrade the QoE of the videos [65]. Finally, additional software plugins (*e.g.*, `CPLEX` [2], TensorFlow [13]) might be required on video clients, which further poses significant barriers for large-scale deployments [30]. The implementation practicality of ABR algorithms severely hinders the exploration of better ABR algorithms.

In response, two categories of possible solutions have been proposed by recent efforts [47, 53, 56, 65]. Nevertheless, these researches fail to achieve high performance and scalability at the same time.

Compromising performance to reduce overhead. Robust-MPC [65] proposed to pre-compute solutions for all network states, construct results into a table, and look up the table when running online. This technique is known as FastMPC. However, the performance degradation brought by the simplification is also drastic. FastMPC



(a) The 10%ile, 50%ile and 90%ile of computation time.

(b) Maximum requesting rate with average response time less than 1s.

Figure 1: Load testing results of remote ABR servers. {R, P, H} refer to {RobustMPC [65], Pensieve [47], HotDASH [53]}. We build ABR servers with `tornado` [6] and test the capacity with `vegeta` [7] from another directly-connected server.

could lead to a performance drop of up to 30%, which is worse than many strawman baselines [65]. Meanwhile, as the solutions are case-specific, ABR designers still have to consider how to simplify and relax their designs with the practicality of online implementation and performance maintenance in mind [17, 65].

Offloading computation to remote ABR servers. Many recent ABR algorithms [14, 47, 53, 65] offload the heavyweight online computation tasks to remote ABR servers. Introducing new servers, however, significantly increases the operating expenses for content providers [40] and thus are not scalable to large-scale deployment. As shown in Figure 1, due to high computation complexity (Figure 1(a)), the ABR server capacity ranges from 20-1000 requests/second/core (Figure 1(b)). However, there might be up to millions of concurrent connections for even one streaming video [5, 30], let alone billions of videos provided by content providers on the Internet [9]. Thus, introducing remote ABR servers may increase the OPEX by up to millions of dollars [§4.4.1] for content providers, which makes the implementation of those algorithms impractical in large-scale deployments.

Employing more complex algorithms in future ABR designs will make the matter worse. Thus, we are motivated to find a *general*, *high-performance*, and *scalable* method to practically implement sophisticated ABR algorithms onto client-side video players, which could relieve ABR designers from considering the practicality issues and make them focus on optimizing the performance of ABR algorithms.

3 PITREE DESIGN

In this section, we first discuss our design choices and challenges on using decision tree in PiTree [§3.1]. We then present the decision tree training algorithm based on imitation learning [§3.2] and provide a theoretical analysis on the upper bound of the average optimization loss of PiTree [§3.3].

3.1 Design Choice and Challenge

3.1.1 Design Choice. As introduced in §1, the design goal of PiTree is to faithfully convert sophisticated ABR algorithms into lightweight and efficient models. Candidates for the target model after conversion include linear regression [46, 50], nonlinear regression [31], and policy sketches [59], *etc.* PiTree employs decision trees due to the following reasons.

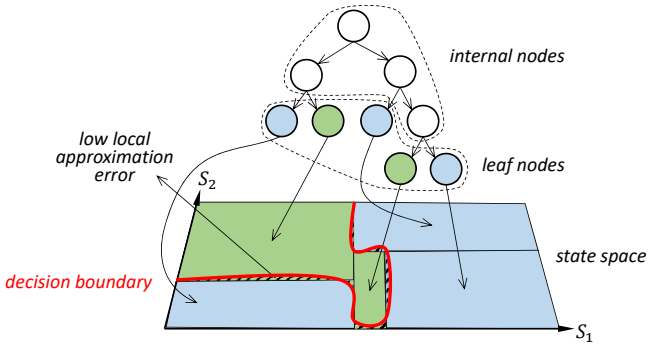


Figure 2: The decision tree can accurately approximate the original decision boundary.

- The rich expressiveness of decision trees enables the high faithfulness of conversion because they are non-parametric and could represent complex policies [20]. As illustrated in Figure 2, decision trees can efficiently approximate the original algorithm even with highly nonlinear decision boundary since they are flexible to scale down to finer granularity when needed.
- Decision trees are lightweight for video players during implementation. Since binary decision trees are comprised of conditional judgments, they could be easily implemented with branching clauses in JavaScript. Our evaluation shows that implementing a decision tree with 100 leaf nodes only increases the page size by <1% (from 381KB to 383KB) [§4.3].
- The structure of decision trees resembles the decision logic of ABR algorithms, which usually contain several judgments from different aspects. For example, optimizing QoE needs to select a high bitrate for the next chunk under the conditions of high current buffer occupancy and network throughput (avoiding rebuffer) and also high current bitrate (ensuring smoothness).

3.1.2 Design Challenge. Since decision tree training is a supervised learning method, it is designed to optimize the loss function (usually the average prediction accuracy [28]) with a large labelled dataset under the distribution of the whole state space [28, 29]:

$$\hat{\pi} = \arg\min_{\pi \in \Pi} \left(\mathbb{E}_{s \sim d_\pi} \left[\mathbb{I}_{\pi(s) \neq a} \right] \right) \tag{1}$$

where d_π is the average distribution of states when using decision tree policy π. $\mathbb{E}$ denotes the expectation over policy π in the set of all policies Π. s and a are the state and action during bitrate adaption. $\mathbb{I}_{\pi(s) \neq a}$ equals to 1 if and only if $\pi(s) \neq a$, and equals to 0 otherwise. However, it is difficult to directly compute the probability distribution of states since the distribution is coupled with traffic throughput, video length, policy preferences, *etc.* Some recent research efforts exhaustively search the action of each state by uniformly sampling in the whole state space [14, 65], which is both inefficient and biased. The dimension of the state space is often high (25 dimensions in Pensieve [47]), and the enumeration of all combinations is inefficient. Meanwhile, since the frequency in the state space might not be distributed uniformly in real world traces, uniformly sampling in the state space might be biased and degrade the performance. In response, we employ the design of a *virtual player* [47, 55] and simulate ABR algorithms with real-world network traces. Compared to packet-level emulations, virtual

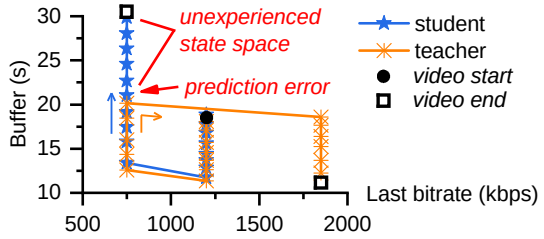


Figure 3: Without imitation learning, one wrong prediction may drive the student off teacher's trajectory in the state space.

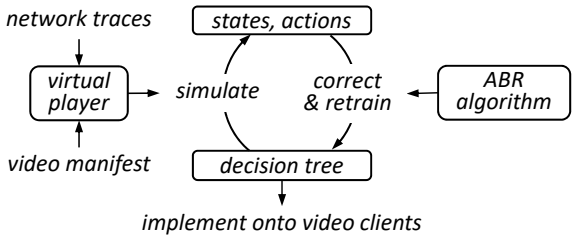


Figure 4: PiTree Overview.

players are fast and efficient since they only calculate chunk-level information. We then collect the state-action pairs during simulations, which are unbiased, since they are generated with real-world traces.

However, faithfully converting ABR algorithms into decision trees with trace-based simulations is also challenging. Due to the cascading effect of ABR algorithms in the video client, even when the prediction is accurate, performance of the converted decision tree might still be low. As shown in Figure 3, a wrong decision may bring the decision tree into a region of unexperienced state space. The decision tree might thus make more mistakes since it has no knowledge about that region of state space. This will further drive the decision tree off the trajectory and worsen the performance. To address this challenge, inspired by recent advances in imitation learning [52], PiTree continuously simulates the decision tree, and lets the original ABR algorithm (teacher) correct the wrong decisions made by that decision tree (student). The decision tree will gradually learn how to make decisions in the whole state space.

3.2 Decision Tree Generation

The overview of PiTree is presented in Figure 4. To convert sophisticated ABR algorithms into decision trees, PiTree uses a *virtual player* [14, 47, 55] to efficiently simulate the dynamics of a real video player and employs imitation learning [16, 35] to improve the faithfulness of the decision tree. With imitation learning, PiTree continuously simulates the performance of the decision tree, and corrects the errors made by the decision tree according to the results of the original ABR algorithm. As shown in Algorithm 1, the imitation learning-based decision tree training algorithm contains the following steps:

Step 1: Initialization. For each ABR algorithm π^*, PiTree first simulates the algorithm in a *virtual player* to collect initial state-action pairs $(\mathbb{S}, \mathbb{A})$ for decision tree training (line 1). The virtual player is a trace-based chunk-level simulator that could precisely mimic the

Algorithm 1: PiTree training based on imitation learning.

Input: ABR Algorithm π^*.
Output: Decision Tree π_M.
1 $(\mathbb{S}, \mathbb{A}) \leftarrow$VirtualPlay($\pi^*$)
2 **foreach** $i \in [1, \cdots, M]$ **do**
3 $\quad \pi_i \leftarrow$TrainDT($\mathbb{S}, \mathbb{A}$)
4 $\quad (\mathbb{S}_i, \mathbb{A}_i) \leftarrow$VirtualPlay($\pi_i$)
5 $\quad \mathbb{A}_i^* \leftarrow$Predict($\pi^*, \mathbb{S}_i$)
6 $\quad$ Aggregate $\mathbb{S} \leftarrow \mathbb{S} \cup \mathbb{S}_i, \mathbb{A} \leftarrow \mathbb{A} \cup \mathbb{A}_i^*$

behaviors of an actual video player with traces and video manifests. For a certain ABR algorithm π^*, the virtual player simulates the algorithm with network traces and video manifests as input. In reality, content providers could use public network traces [10, 51] or collected historical traces [14] for simulation. Moreover, our evaluation shows that PiTree has strong generalization ability if the network traces used at the training phase are statistically different from those in the test environment [§4.4.4].

Specifically, when the ABR algorithm generates a bitrate decision of the following chunk according to current states, the virtual player calculates the states (*e.g.*, rebuffer, download time, *etc.*) at the time of that chunk has been downloaded. The ABR algorithm then takes those states, generates the bitrate decision (action) for the next chunk, and sends the action back to the virtual player. Those state-action pairs are initialized as $(\mathbb{S}, \mathbb{A})$.

Step 2: TrainDT. After initialization, PiTree goes into the imitation learning loop (line 2-6). At the i-th iteration, we first train a decision tree π (student) with current state-action pairs (samples) $(\mathbb{S}, \mathbb{A})$ using Classification and Regression Tree (CART) [28], a well-adopted decision tree training algorithm (line 3). The decision tree takes exactly the same inputs as the original ABR algorithms. Instead of using the 0-1 loss for prediction accuracy (Equation 1), we employ the normalized square loss as the training loss during decision tree generation:

$$\ell(r; r_0) = \frac{(r - r_0)^2}{(R_{max} - R_{min})^2}, \; R_{min} \leqslant r \leqslant R_{max} \tag{2}$$

where $r = \pi(s), r_0 = \pi^*(s)$. s is the current state as introduced in Equation 1. R_{max} and R_{min} are the maximum and minimum bitrates. The intuition behind using the square loss is to penalize student's bitrates that are far from those of the teacher since they have more influence on the playback buffer, *etc.* We also discuss other properties of square-loss function in §3.3. The CART algorithm then greedily splits the samples into leaf nodes to minimize the loss function until either the number of leaf nodes of the decision tree reaches the maximum threshold set by network operators, or all samples have been completely split.

Step 3: VirtualPlay. PiTree then simulates the decision tree π_i in the virtual player and collects a series of new state-action pairs $(\mathbb{S}_i, \mathbb{A}_i)$ (line 4). At this time, although student π_i has already known how to make decisions when facing with the states fed in training, independently simulating π_i might lead to poor performance. As illustrated in Figure 3, due to the cascading effect, many states in $\mathbb{S}_i$ experienced by student π_i in the simulation might not be

experienced during the training in this iteration. We still need to correct the decision tree policy in the following step.

Step 4: Correction. Thus, we feed the states in $\mathbb{S}_i$ to the original ABR algorithm π^* (teacher), and collect the actions $\mathbb{A}_i^*$ made by the teacher (line 5). Finally, we aggregate the student's states and the teacher's actions $(\mathbb{S}_i, \mathbb{A}_i^*)$ with the current state-action pairs $(\mathbb{S}, \mathbb{A})$ (line 6), and go back to ***Step 2*** to continue the next iteration. In this case, when training the decision tree π_{i+1} in the next iteration, it will learn from the mistakes made by the last iteration. The loop continues until it reaches the maximum iteration number (M) set by the user. The decision tree generated by the last iteration will then be implemented into client-side video players.

3.3 Theoretical Analysis

As introduced above, there are two hyper-parameters set by the network operators: the maximum number of iteration M and the maximum number of leaf node N. We discuss the parameter settings of M and N in §4.4.3. In this section, we provide a theoretical analysis on the bound of the average optimization loss of PiTree (unfaithfulness) *w.r.t.* M when implementing the decision tree online. We begin by proving the loss function of PiTree is both Lipschitz [39] and strongly convex:

Theorem 3.1. *$\ell(r; r_0)$ in Equation 2 is both Lipschitz and strongly convex.*

Proof. $\forall r_0, r_1, r_2 \in [R_{min}, R_{max}]$, we have

$$\begin{aligned} |\ell(r_1; r_0) - \ell(r_2; r_0)| &= \frac{|(r_1 - r_0)^2 - (r_2 - r_0)^2|}{(R_{max} - R_{min})^2} \\ &= \frac{|r_1 + r_2 - 2r_0| \cdot |r_1 - r_2|}{(R_{max} - R_{min})^2} \\ &\leqslant \frac{(|r_1 - r_0| + |r_2 - r_0|) \cdot |r_1 - r_2|}{(R_{max} - R_{min})^2} \\ &\leqslant \left(\frac{2}{R_{max} - R_{min}}\right) |r_1 - r_2| \end{aligned} \tag{3}$$

The last inequality holds because $\ell(r; r_0)$ is defined only in $[R_{min}, R_{max}]$. Thus $\ell(r; r_0)$ is Lipschitz with Lipschitz constant $\mathcal{L} = 2/(R_{max} - R_{min})$. Similarly, we could also demonstrate that $\ell(r; r_0)$ is strongly convex (details omitted for brevity): $\forall \lambda \in [0, 1]$, we have:

$$\begin{aligned} &\ell(\lambda r_1 + (1-\lambda) r_2; r_0) \leqslant \\ &\quad \lambda \ell(r_1; r_0) + (1-\lambda)\ell(r_2; r_0) - (\nu/2)\lambda(1-\lambda)(r_1 - r_2)^2 \end{aligned} \tag{4}$$

with strong convexity constant $\nu = 2/(R_{max} - R_{min})^2$. □

Since the loss function $\ell(r; r_0)$ is both Lipschitz and strongly convex on its domain, and also the output actions of ABR algorithms (bitrates) are discrete, we could extend the Theorem 3.3 and Theorem 3.4 introduced in [52] with techniques from [39]. We thus have the following upper bound of the average optimization loss when the decision tree generated by PiTree independently processes videos online:

Theorem 3.2. *For any $\delta > 0$, with training loss ε_M, there exists a policy $\hat{\pi} \in \{\pi_1, \cdots, \pi_M\}$ s.t. the average optimization loss satisfies:*

$$\mathbb{E}_{s \sim d_{\hat{\pi}}} \left[\ell \left(\hat{\pi}(s); \pi^*(s) \right) \right] \leqslant \varepsilon_M + \Theta(1/T) \tag{5}$$

with probability at least $1 - \delta$ as long as $M = \Theta(T \log(1/\delta))$. T is the number of chunks in the video used in the virtual player.

Proof. Let $Q_t^{\pi\prime}(s, \pi)$ denote the t-step cost of executing action a in initial state s and then following policy π':

$$Q_t^{\pi\prime}(s, a) = \left(a - \pi^*(s_1)\right)^2 + \sum_{\tau=2}^{t} \left(\pi'(s_\tau) - \pi^*(s_\tau)\right)^2 \tag{6}$$

where s_τ is the state at the time τ. Thus we have $\forall a, t \in [1, T]$,

$$Q_{T-t+1}^{\pi*}(s, a) - Q_{T-t+1}^{\pi*}\left(s, \pi^*(s)\right) = \frac{(a - \pi^*(s))^2}{(R_{max} - R_{min})^2} \leqslant 1 \triangleq u \tag{7}$$

Hence the proof follows [52] and [39] with the fact that $u = 1$. □

$\hat{\pi}$ could be found by cross-validation among decision trees at different iterations, which is usually the decision tree from the last iteration π_M in our experiments. Thus we provide an upper bound for the average optimization loss of PiTree. The training loss ε_M is related to the complexity of original ABR algorithm and the number of leaf nodes N (expressiveness of decision tree). Moreover, our evaluation demonstrates that PiTree is quite robust to the number of leaf nodes [§4.4.3].

4 EVALUATION

We apply PiTree over three representative ABR algorithms [47, 53, 65], with three network traces, and on three QoE metrics. We introduce implementation details in §4.1 and evaluate PiTree by answering the following questions:

- ***Performance.*** Can PiTree faithfully convert sophisticated ABR algorithms into decision trees? Our evaluation results demonstrate that the average performance degradation caused by PiTree is within 3% for all three ABR algorithms [§4.2].
- ***Overhead.*** How much resources will PiTree consume when algorithms are executed online? We demonstrate that the page size, decision-making latency, and runtime memory utilization of PiTree-based methods are reduced significantly compared to the original ones [§4.3].
- ***Deployment Efforts.*** Do network operators need to pay many additional efforts to deploy PiTree in practice? Our evaluation shows that PiTree , PiTree could save considerable operating expenses, consume acceptable additional offline training time, and have robust parameter settings and strong generalization ability [§4.4].

4.1 Experiment Setup

4.1.1 Video Sample. We evaluate PiTree with "EnvivoDash3" video from the MPEG-DASH reference videos with a length of 193 seconds, which has been used in prior work [14, 47]. The video is partitioned into 4-second chunks with bitrates of {300, 750, 1200, 1850, 2850, 4300} kbps.

4.1.2 QoE Metrics. The QoE metric can be expressed as:

$$QoE = \sum_n q(R_n) - \mu \sum_n T_n - \sum_n |q(R_{n+1}) - q(R_n)| \tag{8}$$

where R_n represents the bitrate of chunk n. T_n is the rebuffering time of chunk n. $q(\cdot)$ is the utilization function as defined in Table 2. To better illustrate the individual performance of different parts

Table 2: QoE metrics considered in our evaluation [47, 64].

QoE_{lin}	$q(R) = R$	μ=4.3
QoE_{log}	$q(R) = \log(R/R_{min})$	μ=2.66
QoE_{hd}	$q(R) = \{0.3{:}1, 0.75{:}2, 1.2{:}3, 1.85{:}12, 2.85{:}15, 4.3{:}20\}$	μ=8

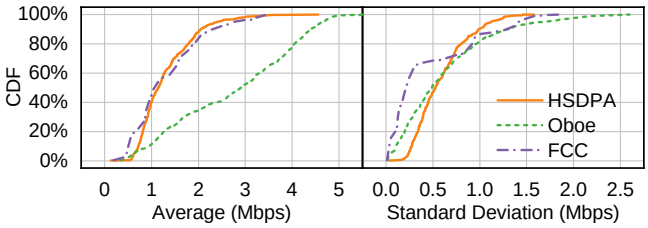


Figure 5: The statistics of the throughput of network traces.

of QoE, we consider three choices of $q(R_n)$ in prior work [47, 56, 65]. Three terms in Equation 8 respectively refer to video quality, rebuffer penalty and smoothness penalty.

4.1.3 Network Traces. As for network traces, to make a fair comparison, we adopt the traces used in the evaluation of previous work [14, 47, 65]. We use 460 traces from Norway's 3G HSDPA [51], 264 traces from US FCC broadband [10] compiled by [47], and 428 traces from Oboe [14] to evaluate PiTree. These three sets of traces are statistically different as presented in Figure 5.

We apply PiTree over the following state-of-the-art ABR algorithms[1]:

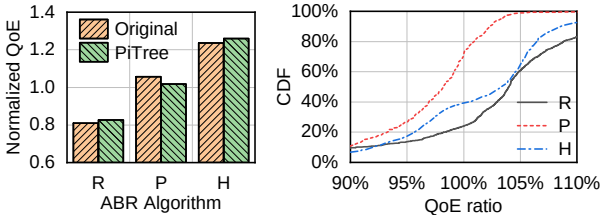
- **RobustMPC** [65] employs Mixed Integer Linear Programming to calculate optimal bitrate over future chunks with buffer occupancy and network throughput.
- **Pensieve** [47] models the bitrate selection process with Reinforcement Learning (RL) and makes predictions based on 25 states with a neural network.
- **HotDASH** [53] extends Pensieve and uses two cascaded neural networks to make ABR decisions.

4.1.4 Testbed Setup. We use the virtual player in [47] for decision tree training. We migrate the decision tree generated by PiTree into `dash.js` [11] and compress JavaScript codes with the `UglifyJS` plugin in the `Grunt.js` [12]. The video server and emulation method [48] are the same with those in Pensieve [47]. We leave the large-scale real-world test for future work. Due to the difference of the complexity of ABR algorithms, we set the number of leaf nodes to 500, 100 and 100 for RobustMPC, Pensieve and HotDASH. We discuss this setting in §4.4.3.

4.2 Performance

We demonstrate the performance maintenance of PiTree by comparing the QoE of original algorithms and decision trees converted with PiTree. We thus measure the ratio of QoE by the PiTree-generated decision trees and the original algorithms. A QoE ratio less than 100% indicates a performance degradation. We first measure the average QoE normalized by the number of chunks and average QoE ratio across three types of QoE metrics and three sets of traces, as shown in Figure 6. The average performance degradation is less

[1]RobustMPC and Pensieve from https://github.com/hongzimao/pensieve, HotDASH from https://github.com/SatadalSengupta/hotdash.



(a) Normalized QoE. (b) QoE ratio.

Figure 6: Overall average results of PiTree. {R, P, H} refer to {RobustMPC, Pensieve, HotDASH} respectively.

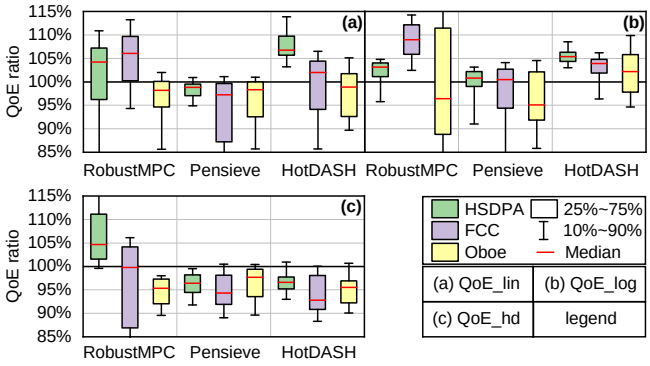


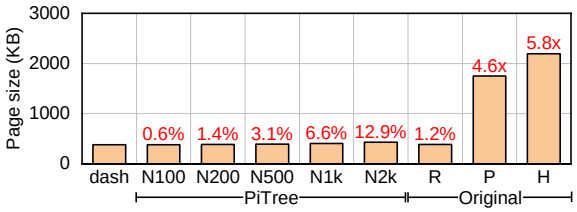
Figure 7: QoE ratio of PiTree on different ABR algorithms, network traces and QoE metrics.

than 3% for three algorithms (average QoE ratio of Pensieve is 97% as presented in Figure 6(b)), which is negligible compared to the performance improvement achieved by new algorithms (17% for HotDASH over Pensieve, and 30% for Pensieve over RobustMPC). Detailed results on different traces and different QoE metrics are presented in Figure 7. Most of the median performance degradation is less than 5%, which demonstrates that PiTree could faithfully convert the sophisticated algorithm across a wide range of scenarios. PiTree can also accurately imitate the behavior of original ABR algorithms at individual bitrate level. The individual prediction accuracy for RobustMPC, Pensieve and HotDASH are 80%, 90% and 82% respectively, more details of which are presented in Figure 11 in §4.4.3.

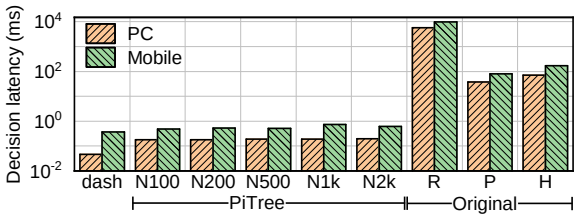
4.3 Overhead

We measure the overhead of implementing PiTree into video players across several metrics with different numbers of leaf nodes. As decision trees converted from different algorithms have similar overhead, we present the average results of decision trees with three sets of traces and three ABR algorithms.

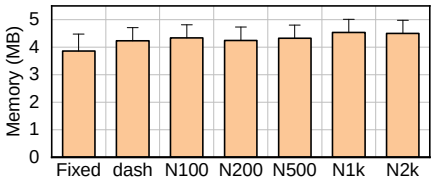
4.3.1 Page Size. We first measure the HTML page size and present the results in Figure 8(a). We also implement original ABR algorithms into video players and demonstrate their impracticality. `dash` represents the rate-based algorithm adopted in `dash.js`. Compared to the original `dash`-based page, Pensieve increases the page size by 4.6× (from 381KB to 1750KB) with `Tensorflow.js` [54]. This drastically increases the page load time by 10s when the goodput is 1200kbps. Users have to wait for a long time before the video can



(a) Page size.



(b) Decision-making latency.



(c) Runtime memory. The error bar represents the peak value of memory utilization.

Figure 8: Overhead of PiTree. `N100`: the decision tree with 100 leaf nodes. {R, P, H}: {RobustMPC, Pensieve, HotDASH}.

play, which might drive some of them to leave the page [41]. In contrast, results show that even with 2000 leaf nodes (N2k), the page size of PiTree is only increased by 13%. Moreover, our experiments in §4.2 demonstrate that decision trees with 100 leaf nodes are faithful enough for Pensieve and HotDASH, which only increases the page size by about 0.6%. This only introduces a negligible additional page load time by 0.01s.

4.3.2 Decision Latency. We further measure the decision latency within JavaScript of PiTree-based ABR decision trees and the original rate-based ABR algorithm in `dash.js`. Since the decision-making latency is highly related to underlying devices, we measure the latency on two testbeds: a PC with an Intel Core i7-8550 CPU, and a mobile phone with a Qualcomm Snapdragon 821 CPU. As shown in Figure 8(b), the decision latency of original algorithms is found to be 1s, 3-4 magnitudes larger than that of PiTree-based algorithms. Such a high decision latency will not only impair the QoE due to the out-of-dated information [65], but will also stall the video player when the video chunk length is less than the average decision latency (*e.g.*, 2s in [38]). In contrast, the average decision-making latency of PiTree is significantly reduced to less than 1ms, which is at the same magnitude with the default ABR algorithm in `dash.js`.

4.3.3 Memory Utilization. We finally measure the average runtime JavaScript heap memory with the `memory` API in Chrome DevTools [1]. We implement a *fixed bitrate algorithm* as a baseline, which constantly selects the lowest bitrate, to eliminate the influence from other functions in the video player. As shown in

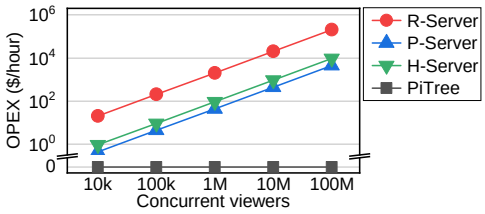


Figure 9: Estimated operating expenses of ABR servers.

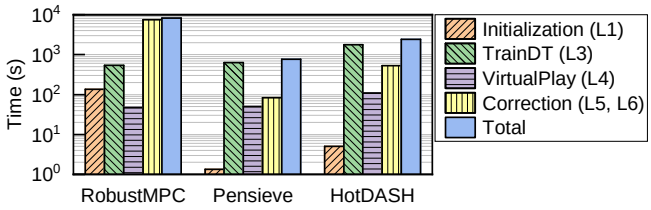


Figure 10: Offline training time breakdown.

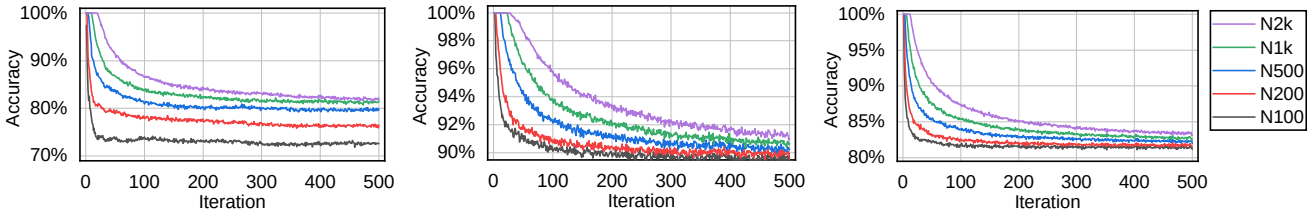
Figure 8(c), the average runtime memory is increased by less than 7% for all decision trees, which is negligible compared to other components in the video player.

4.4 Deployment Efforts

4.4.1 Operating Expenses. We further compare the OPEX of PiTree with other server-based ABR solutions. With the ABR server capacity measured in Figure 1(b), we calculate the OPEX per hour for three ABR algorithms based on the server operating expenses. For example, a 4-core Amazon EC2 instance with similar configurations costs $0.188 per hour (`t2.xlarge`) [3, 40]. As shown in Figure 9, for large content providers such as YouTube [8], with more than one billion hours of video clips being watched daily on average [9], the average concurrent viewer is approximately $\frac{1Bh}{24h} = 40M$. Thus they need to pay up to millions of dollars monthly for remote ABR servers. Although the estimation here is an extreme case in the real world, it is indisputable that reducing considerable online servers will save OPEX for content providers. This cost makes the server-based ABR solutions not scalable. In contrast, since PiTree-based solutions are directly implemented into video clients, they do not introduce additional OPEX and thus prevent revenue loss for large-scale content providers.

4.4.2 Offline Training Cost. We break down the training time for each algorithm for 500 iterations and present the results in Figure 10 according to steps in Algorithm 1. The time of `TrainDT` and `VirtualPlay` does not vary much with respect to ABR algorithms. Note that the predictions in line 5 in Algorithm 1 are accelerated by 32 parallel virtual CPU cores, which again demonstrates the complexity of state-of-the-art ABR algorithms. The total training time is up to 2.3 hours for three ABR algorithms, which is acceptable since it needs running offline only once for initial decision tree generation at the design phase.

4.4.3 Sensitivity Analysis. To test the sensitivity of the number of leaf nodes in PiTree, we vary the number of leaf nodes from 100 to 2000 and measure the single prediction accuracy for the three ABR algorithms evaluated before. Results are presented in Figure 11. For RobustMPC, as shown in Figure 11(a), the accuracy of decision trees



(a) RobustMPC. (b) Pensieve. (c) HotDASH.

Figure 11: Single prediction accuracy of PiTree at different iterations. The accuracy goes down because $(\mathbb{S}, \mathbb{A})$ lacks samples in the first several iterations and decision trees are overfitted.

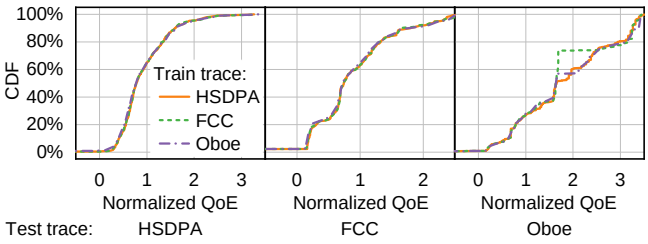


Figure 12: Normalized QoE_{lin} of PiTree-over-Pensieve when test traces are statistically different from training traces.

with less than 500 leaf nodes converge to different levels since with better expressiveness, the performance will be improved. However, the improvement is not unlimited. As long as the number of leaf nodes is high enough to express the policy of RobustMPC, more nodes will lead to overfitting and thus a slower convergence (*e.g.*, N1k and N2k in Figure 11(a)). Decision trees with different numbers of leaf nodes for Pensieve and HotDASH (Figure 11(b), 11(c)) demonstrate the similar relationship. The converged performance of PiTree is hardly affected as long as the number of leaf nodes is above a certain level, indicating a strong robustness towards the number of leaf nodes.

4.4.4 Generalization of PiTree. In previous experiments, we randomly split a set of traces into training (80%) and test (20%) set for evaluation. We want to further investigate the generalization ability of PiTree when the statistical features of the training and test set are different. We measure the normalized QoE_{lin} of the decision trees converted from Pensieve but trained with different traces in different test traces. As shown in Figure 12, even decision trees are trained with different traces, they perform almost the same during testing. We also measure the ratio of the normalized QoE over that when the training and test environments are the same, the median of which in all experiments in Figure 12 is greater than 97%. Performance over Oboe on some certain traces are degraded a little since the traffic distribution of Oboe is quite different from those of the other two sets of traces (Figure 5). Experiment results for other ABR algorithms and QoE metrics are similar (not presented), which demonstrate the strong generalization ability of PiTree. Moreover, advanced online parameter tuning techniques [14, 26] could be employed to further enhance the generalization ability. We leave the generalization ability over large-scale real-world deployments as our future work.

5 RELATED WORK

ABR Algorithms. As summarized in Table 1, recent research efforts include buffer-based methods [34, 55, 56], rate-based methods [38, 44, 61], conventional hybrid methods [25, 60, 63, 65], and ML-based hybrid methods [32, 45, 47, 53]. All of them could be converted into decision trees with PiTree if they are too heavyweight for large-scale real world implementations in practice. There are also recent efforts on the co-design of video server, network protocol and video player in ABR systems [18, 19, 37], and also online parameter finetuning mechanisms of ABR algorithms [14, 26]. Both of them are orthogonal to PiTree and could be integrated together for further improvements.

Complex Algorithm Deployment. There are also some recent work on how to deploy sophisticated models in practice, most of which focus on heavyweight neural networks. Proposed methods include introducing new acceleration devices [23, 42, 64], or compressing neural networks [21, 22, 43]. However, most of them are expensive, case-specific and difficult to be generalized to other methods. In contrast, as PiTree does not require any information from the ABR algorithms, it could be applied to any sophisticated algorithms. Moreover, instead of introducing new expenses [23, 42, 64], the online overhead and deployment efforts of PiTree are negligible [§4.3, §4.4].

6 CONCLUSIONS

In this paper, we propose PiTree, a new framework to generally make the implementation of sophisticated ABR algorithms practical in the real world. PiTree faithfully converts different ABR algorithms into decision trees with the help of offline imitation learning with theoretically bounded average optimization loss. Evaluations over three representative ABR algorithms show that PiTree could achieve high performance, low runtime overhead at the same time with negligible additional deployment efforts. We believe that PiTree could accelerate the design of new ABR algorithms.

ACKNOWLEDGEMENT

We sincerely thank Minhu Wang, Yuan Yang, anonymous MM'19 reviewers, and our shepherd Wei Tsang Ooi for their valuable suggestions on this paper. This paper is supported by National Key R&D Program of China (2017YFB0801701) and National Science Foundation of China (61625203, 61832013, 61872426). Mingwei Xu is the corresponding author.

REFERENCES

[1] Chrome devtools. https://developers.google.com/web/tools/chrome-devtools/.
[2] Cplex optimizer. https://www.ibm.com/analytics/cplex-optimizer.
[3] Ec2 instance pricing – amazon web services (aws). https://aws.amazon.com/ec2/pricing/reserved-instances/pricing/.
[4] Mobile accounted for 62 percent of online video views. https://www.statista.com/statistics/444318/mobile-device-video-views-share/.
[5] Official youtube blog: With nearly 2 million concurrent viewers and over 3 million live watch hours, first presidential debate breaks political record. https://youtube.googleblog.com/2016/09/with-nearly-2-million-concurrent.html.
[6] Tornado web server. https://www.tornadoweb.org/.
[7] tsenart/vegeta: Http load testing tool and library. it's over 9000! https://github.com/tsenart/vegeta.
[8] Youtube. https://www.youtube.com/.
[9] Youtube revenue and usage statistics (2018) - business of apps. http://www.businessofapps.com/data/youtube-statistics/.
[10] Raw data - measuring broadband america. https://www.fcc.gov/reports-research/reports/measuring-broadband-america/raw-data-measuring-broadband-america-2016, 2016.
[11] Dash.js. https://github.com/Dash-Industry-Forum/dash.js, 2018.
[12] gruntjs/grunt-contrib-uglify: Minify files with uglifyjs. https://github.com/gruntjs/grunt-contrib-uglify, 2019.
[13] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., et al. Tensorflow: A system for large-scale machine learning. In *USENIX OSDI* (2016).
[14] Akhtar, Z., Nam, Y. S., Govindan, R., Rao, S., Chen, J., Katz-Bassett, E., Ribeiro, B., Zhan, J., and Zhang, H. Oboe: auto-tuning video abr algorithms to network conditions. In *ACM SIGCOMM* (2018).
[15] Ba, J., and Caruana, R. Do deep nets really need to be deep? In *NIPS* (2014).
[16] Bastani, O., Pu, Y., and Solar-Lezama, A. Verifiable reinforcement learning via policy extraction. In *NeurIPS* (2018).
[17] Beben, A., Wiśniewski, P., Batalla, J. M., and Krawiec, P. Abma+: lightweight and efficient algorithm for http adaptive streaming. In *ACM MMSys* (2016).
[18] Ben Mustafa, I., Nadeem, T., and Halepovic, E. Flexstream: Towards flexible adaptive video streaming on end devices using extreme sdn. In *ACM MM* (2018), pp. 555–563.
[19] Bentaleb, A., Begen, A. C., Harous, S., and Zimmermann, R. A distributed approach for bitrate selection in http adaptive streaming. In *ACM MM* (2018), pp. 573–581.
[20] Blockeel, H., and De Raedt, L. Top-down induction of first-order logical decision trees. *Artificial intelligence 101*, 1-2 (1998), 285–297.
[21] Bucilua, C., Caruana, R., and Niculescu-Mizil, A. Model compression. In *ACM KDD* (2006).
[22] Chen, W., Wilson, J., Tyree, S., Weinberger, K., and Chen, Y. Compressing neural networks with the hashing trick. In *ICML* (2015).
[23] Chen, Y.-H., Krishna, T., Emer, J. S., and Sze, V. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits 52*, 1 (2017), 127–138.
[24] Cui, Y., Ooi, W. T., Liu, J., Zhang, X., Bentaleb, A., Zheng, K., and Li, Y. Acm multimedia 2019 grand challenge - live video streaming. https://www.aitrans.online/MMGC/, 2019.
[25] De Cicco, L., Caldaralo, V., Palmisano, V., and Mascolo, S. Elastic: A client-side controller for dynamic adaptive streaming over http (dash). In *IEEE International Packet Video Workshop* (2013).
[26] De Cicco, L., Cilli, G., and Mascolo, S. Erudite: a deep neural network for optimal tuning of adaptive video streaming controllers. In *ACM MMSys* (2019).
[27] Dobrian, F., Sekar, V., Awan, A., Stoica, I., Joseph, D., Ganjam, A., Zhan, J., and Zhang, H. Understanding the impact of video quality on user engagement. In *ACM SIGCOMM* (2011).
[28] Friedman, J. H., Olshen, R. A., Stone, C. J., et al. Classification and regression trees. *Belmont, CA: Wadsworth & Brooks* (1984).
[29] Gama, J., Rocha, R., and Medas, P. Accurate decision trees for mining high-speed data streams. In *ACM KDD* (2003).
[30] Ganjam, A., Siddiqui, F., Zhan, J., Liu, X., Stoica, I., Jiang, J., Sekar, V., and Zhang, H. C3: Internet-scale control plane for video quality optimization. In *USENIX NSDI* (2015), pp. 131–144.
[31] Guo, W., Mu, D., Xu, J., Su, P., Wang, G., and Xing, X. Lemna: Explaining deep learning based security applications. In *ACM CCS* (2018).
[32] Huang, T., Yao, X., Wu, C., Zhang, R.-X., and Sun, L. Tiyuntsong: A self-play reinforcement learning approach for abr video streaming. *arXiv:1811.06166* (2018).
[33] Huang, T.-Y., Handigol, N., Heller, B., McKeown, N., and Johari, R. Confused, timid, and unstable: Picking a video streaming rate is hard. In *ACM IMC* (2012), pp. 225–238.
[34] Huang, T.-Y., Johari, R., McKeown, N., Trunnell, M., and Watson, M. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. In *ACM SIGCOMM* (2014).
[35] Hussein, A., Gaber, M. M., Elyan, E., and Jayne, C. Imitation learning: A survey of learning methods. *ACM Comput. Surv. 50*, 2 (Apr. 2017), 21:1–21:35.
[36] Index, C. V. N. Forecast and methodology, 2016–2021. *White Paper, June* (2017).
[37] Jiang, J., Sekar, V., Milner, H., Shepherd, D., Stoica, I., and Zhang, H. Cfa: A practical prediction system for video qoe optimization. In *USENIX NSDI* (2016), pp. 137–150.
[38] Jiang, J., Sekar, V., and Zhang, H. Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive. In *ACM CoNEXT* (2012), pp. 97–108.
[39] Kakade, S. M., and Tewari, A. On the generalization ability of online strongly convex programming algorithms. In *NIPS* (2008).
[40] Koller, R., and Williams, D. Will serverless end the dominance of linux in the cloud? In *ACM HotOS* (2017), pp. 169–173.
[41] Krishnan, S. S., and Sitaraman, R. K. Video stream quality impacts viewer behavior: Inferring causality using quasi-experimental designs. In *ACM IMC* (2012), pp. 211–224.
[42] Lane, N. D., Georgiev, P., and Qendro, L. Deepear: Robust smartphone audio sensing in unconstrained acoustic environments using deep learning. In *ACM Ubicomp* (2015).
[43] Li, H., Kadav, A., Durdanovic, I., Samet, H., and Graf, H. P. Pruning filters for efficient convnets. In *ICLR* (2017).
[44] Li, Z., Zhu, X., Gahm, J., Pan, R., Hu, H., Begen, A. C., and Oran, D. Probe and adapt: Rate adaptation for http video streaming at scale. *IEEE Journal on Selected Areas in Communications 32*, 4 (2014), 719–733.
[45] Liu, J., Tao, X., and Lu, J. Qoe-oriented rate adaptation for dash with enhanced deep q-learning. *IEEE Access 7* (2019), 8454–8469.
[46] Lundberg, S. M., and Lee, S.-I. A unified approach to interpreting model predictions. In *NIPS* (2017).
[47] Mao, H., Netravali, R., and Alizadeh, M. Neural adaptive video streaming with pensieve. In *ACM SIGCOMM* (2017), pp. 197–210.
[48] Netravali, R., Sivaraman, A., Das, S., Goyal, A., Winstein, K., Mickens, J., and Balakrishnan, H. Mahimahi: Accurate record-and-replay for HTTP. In *USENIX ATC* (2015), pp. 417–429.
[49] Ramos-Muñoz, J. J., Prados-Garzon, J., Ameigeiras, P., Navarro-Ortiz, J., and López-Soler, J. M. Characteristics of mobile youtube traffic. *IEEE Wireless Communications 21*, 1 (2014), 18–25.
[50] Ribeiro, M. T., Singh, S., and Guestrin, C. Why should i trust you?: Explaining the predictions of any classifier. In *ACM KDD* (2016).
[51] Riiser, H., Vigmostad, P., Griwodz, C., and Halvorsen, P. Commute path bandwidth traces from 3g networks: Analysis and applications. In *ACM MMSys* (2013), pp. 114–118.
[52] Ross, S., Gordon, G., and Bagnell, D. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS* (2011).
[53] Sengupta, S., Ganguly, N., Chakraborty, S., and De, P. Hotdash: Hotspot aware adaptive video streaming using deep reinforcement learning. In *IEEE ICNP* (2018), pp. 165–175.
[54] Smilkov, D., Thorat, N., Assogba, Y., Nicholson, C., Kreeger, N., Yu, P., Cai, S., Nielsen, E., et al. Tensorflow.js: Machine learning for the web and beyond (https://js.tensorflow.org/). In *SysML* (2019).
[55] Spiteri, K., Sitaraman, R., and Sparacio, D. From theory to practice: improving bitrate adaptation in the dash reference player. In *ACM MMSys* (2018).
[56] Spiteri, K., Urgaonkar, R., and Sitaraman, R. K. Bola: Near-optimal bitrate adaptation for online videos. In *IEEE INFOCOM* (2016).
[57] Stockhammer, T. Dynamic adaptive streaming over http – standards and design principles. In *ACM MMSys* (2011), pp. 133–144.
[58] Sun, Y., Yin, X., Jiang, J., Sekar, V., Lin, F., Wang, N., Liu, T., and Sinopoli, B. Cs2p: Improving video bitrate selection and adaptation with data-driven throughput prediction. In *ACM SIGCOMM* (2016).
[59] Verma, A., Murali, V., Singh, R., Kohli, P., and Chaudhuri, S. Programmatically interpretable reinforcement learning. In *ICML* (2018).
[60] Wang, B., and Ren, F. Towards forward-looking online bitrate adaptation for dash. In *ACM MM* (2017), pp. 1122–1129.
[61] Wang, C., Rizk, A., and Zink, M. Squad: A spectrum-based quality adaptation for dynamic adaptive streaming over http. In *ACM MMSys* (2016).
[62] Wu, J., Cheng, B., Yuen, C., Shang, Y., and Chen, J. Distortion-aware concurrent multipath transfer for mobile video streaming in heterogeneous wireless networks. *IEEE Transactions on Mobile Computing 14*, 4 (2015), 688–701.
[63] Yadav, P. K., Shafiei, A., and Ooi, W. T. Quetra: A queuing theory approach to dash rate adaptation. In *ACM MM* (2017).
[64] Yeo, H., Jung, Y., Kim, J., Shin, J., and Han, D. Neural adaptive content-aware internet video delivery. In *USENIX OSDI* (2018).
[65] Yin, X., Jindal, A., Sekar, V., and Sinopoli, B. A control-theoretic approach for dynamic adaptive video streaming over http. In *ACM SIGCOMM* (2015).
[66] Zheng, Y., Liu, Z., You, X., Xu, Y., and Jiang, J. Demystifying deep learning in networking. In *ACM APNet* (2018).