Practically Deploying Heavyweight Adaptive Bitrate Algorithms With Teacher-Student Learning

Zili Meng, Yaning Guo, Yixin Shen, Jing Chen, Chao Zhou, Minhu Wang, Jia Zhang, Mingwei Xu, Chen Sun, Hongxin Hu

***Abstract*—Major commercial client-side video players employ adaptive bitrate (ABR) algorithms to improve the 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 deployed in client devices with limited resources, such as mobile phones. Existing solutions suffer from a trade-off between algorithm performance and deployment overhead. To make the deployment of sophisticated ABR algorithms practical, we propose PiTree, a *general*, *high-performance*, and *scalable* framework that can faithfully convert sophisticated ABR algorithms into decision trees with teacher-student learning. In this way, network operators can train complex models offline and deploy converted lightweight decision trees online. We also present theoretical analysis on the conversion and provide two upper bounds of the prediction error during the conversion and the generalization loss after conversion. Evaluation on three representative ABR algorithms with both trace-driven emulation and real-world experiments demonstrates that PiTree could convert ABR algorithms into decision trees with <3% average performance degradation. Moreover, compared to original deployment solutions, PiTree could save considerable operating expenses for content providers.**

***Index Terms*—Adaptive bitrate streaming, practicality, client-side implementation, decision tree.**

I. Introduction

In recent years, video streaming traffic plays a prominent role in Internet traffic [2]. Meanwhile, online video clients have increasingly higher demands on the video quality of experience (QoE) [3], which directly correlates with content provider revenue [4]. Therefore, as presented in Table I, a series of adaptive bitrate (ABR) algorithms are proposed to optimize the video quality, some of which have already been used by commercial content providers [5, 6]. These algorithms usually run on client-side video players that dynamically select a bitrate based on network conditions. ABR algorithms have to handle complicated situations, including different QoE demands [5, 7], high variation of network throughput [3], and the cascading effect between actions [3]. Therefore, sophisticated algorithms (e.g., Integer Programming [8], Lyapunov optimization [5], and neural networks [3, 7, 9]) are adopted to improve ABR performance.

Table I
Representative ABR Algorithms in recent years.

Publication Year: 2012			2019
Rate-based	Festive [11]	Panda [12]	Squad [13]
Buffer-based	BBA [14]	BOLA [5]	BOLA-E [6]
Hybrid (Non-ML)	Elastic [15]	RobustMPC [8]	MSPC [16]
Hybrid (ML)	Pensieve [3]	HotDASH [9]	Comyco [17]

However, the expensive computation overhead of increasingly complex ABR algorithms prevents them from traditional in-player implementations [3, 8]. Notably, a sharply increasing number of users choose to play videos through smart TVs and mobile devices such as pads or cellphones [9]. The latest statistics indicate that mobile devices account for 62% of online video views in 2018, and this number is increasing rapidly [2]. These mobile devices often have very limited computation resources, which cannot satisfy the resource requirements of solving complex optimization problems (§II-B). Thus, it is difficult to directly integrate ABR algorithms into HTML pages and implement them in client-side players [10]. The situation will become worse when pursuing higher performance with more complex optimizations in future.

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 (§II-B): (*i*) *Compromising performance to reduce overhead*. RobustMPC [8] provides an online version of its integer programming-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 drastically degrades the performance even below many strawman baselines [8]. (*ii*) *Offloading computation to remote ABR servers*. Many recent efforts [3, 9] suggest offloading the heavyweight computation to remote ABR servers, requiring video clients to send requests to the ABR servers when they need to make

Manuscript received February 25, 2020; revised August 22, 2020, October 27, 2020; accepted December 13, 2020; approved by IEEE/ACM Transactions on Networking Editor S. Rao. Date of publication XX, 2020; date of current version XX. The research is supported by the National Natural Science Foundation of China (No. 61625203 and 61832013), and the National Key R&D Program of China (No. 2017YFB0801701). A previous version of this work [1] has been presented in ACM International Conference on Multimedia 2019 (MM'19) (*Corresponding author: Mingwei Xu*).

Z. Meng, Y. Guo, Y. Shen, J. Chen, M. Wang, and C. Sun are with the Institute for Network Sciences and Cyberspace, Tsinghua University (e-mail: zilim@ieee.org; {gyn17, shen-yx18, j-chen20, wangmh19}@mails.tsinghua.edu.cn; c-sun14@tsinghua.org.cn)

C. Zhou is with Beijing Kuaishou Technology Co., Ltd. (email: zhouchaoyf@gmail.com).

J. Zhang and M. Xu are with the Institute for Network Sciences and Cyberspace, Tsinghua University, also with the Department of Computer Science and Technology, Tsinghua University, also with the Beijing National Research Center for Information Science and Technology (email: jiazhan18@mails.tsinghua.edu.cn, xumw@tsinghua.edu.cn).

H. Hu is with University at Buffalo, the State University of New York (e-mail: hongxinh@buffalo.edu).

Digital Object Identifier 10.1109/XX.1234

decisions. The content providers, however, have to introduce and maintain additional servers to provide ABR optimization services specifically, which significantly increases operating expenses (OPEX). Therefore, we are motivated to enable the large-scale deployment of heavyweight ABR algorithms.

Inspired by recent advances in interpreting complicated models [18–20], our key observation is that the offline design and online deployment of ABR algorithms can be optimized separately. The reason behind this observation is because some lightweight representations of algorithms (*e.g.*, decision trees) have similar expressiveness compared to those complex representations [20]. Therefore, we could implement sophisticated ABR algorithms in practice by decoupling offline design and online deployment. Algorithm developers can still design and train any ABR algorithms as they wish, and we can *convert* the finetuned models into other lightweight representations to reduce the online computation overhead.

Based on this observation, we propose PiTree, a *general*, *high-performance*, and *scalable* framework to bridge the gap between offline design and online deployment of ABR algorithms using *decision trees*. The key idea of PiTree is to convert sophisticated offline algorithms into lightweight online decision trees to achieve high performance and implementation scalability simultaneously. Due to the complex decision space of ABR algorithms (§III-A) and the cascading effects between actions [3], however, we are challenged to faithfully and efficiently convert sophisticated ABR algorithms into decision trees. In response, PiTree adopts recent advances in the machine learning community, namely *teacher-student learning* [20], to faithfully convert ABR algorithms to decision trees with negligible performance degradation. The original ABR algorithm acts as a teacher who continuously *corrects* the actions of the student decision tree. Moreover, to make network operators confident about the online deployment of decision trees, we provide a theoretical analysis of PiTree and guarantee the upper bound of the optimization loss and generalization loss of the performance.

We implement PiTree over three state-of-the-art ABR algorithms (RobustMPC [8], Pensieve [3], and HotDASH [9]). We evaluate the performance of PiTree with both existing datasets and new traces and videos collected by us, together with several days of real-world experiments. Our evaluation shows that the performance degradation of the decision tree against original ABR algorithms is maintained within 3% on different network traces (§VI-A). More importantly, PiTree could reduce the HTML page size by around 10× and decision latency by up to three magnitudes, and save considerable operating expenses (§VI-B). Further evaluation results also demonstrate the robustness and generalization ability of PiTree on various traces with different settings (§VI-C).

In summary, we make the following contributions:

- We illustrate the problem of practically deploying sophisticated ABR algorithms for content providers and motivate the problem with concrete server-based experiments.
- We present PiTree, a general, high-performance, and scalable framework for the practical deployment of ABR algorithms using decision trees.
- We analyze the conversion procedure of PiTree and provide two theoretical upper bounds of the prediction error and generalization loss of PiTree during the conversion.
- We evaluate PiTree with emulations and real-world implementations and demonstrate that PiTree could convert state-of-the-art ABR algorithms into lightweight decision trees with negligible performance degradation.

To the best of our knowledge, PiTree is the first general framework to make it practical to deploy state-of-the-art sophisticated ABR algorithms on client-side video players. The source codes and datasets of PiTree are available at https://transys.io/pitree/. We believe that PiTree will accelerate the deployment of new sophisticated ABR algorithms.

Roadmap. We introduce the background and motivations and articulate our design choice of PiTree in §II. The decision tree generation algorithm is presented in §III. We then propose theoretical bounds for the performance of PiTree in §IV and introduce the implementations of PiTree with our dataset in §V. Evaluation results with trace-driven emulation and real-world experiments are presented in §VI. We further discuss the potential directions and limitations of PiTree in §VII and introduce the related research efforts in §VIII.

II. Background and Motivation

In this section, we first introduce the background of ABR streaming (§II-A), then motivate the design of PiTree with illustrative experiments (§II-B).

A. ABR Streaming

Dynamic Adaptive Streaming over HTTP (DASH) [10] is the predominant method for streaming video delivery today. In DASH systems, each video is partitioned into chunks (e.g., 4-second blocks) and each video 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. It is well-established that a higher QoE follows from (*i*) higher average video bitrate, (*ii*) fewer rebuffering events, and (*iii*) better video bitrate smoothness [3, 7, 9]. 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 [3]. These factors make the optimization of QoE challenging.

Existing solutions have already achieved significant improvements in addressing the conflicts above. As summarized in Table I, recent research efforts include buffer-based methods [5, 14], which make decisions based on video player buffer occupancy, rate-based methods [12, 13], which make decisions with network throughput information, and hybrid methods [8, 15], utilizing observations from network and playback buffer. Especially with the rapid development of

machine learning, many recent algorithms employ neural networks [3, 9] to further improve the QoE. However, since there still exists a gap between current ABR algorithms and the offline optimal solutions [3, 4], further efforts on ABR algorithms are still needed for better performance. As an example, the academic community continuously holds competition seeking better QoE algorithms [21]. In summary, increasingly optimized methods have been (and are going to be) proposed for better performance in the competition of QoE [21].

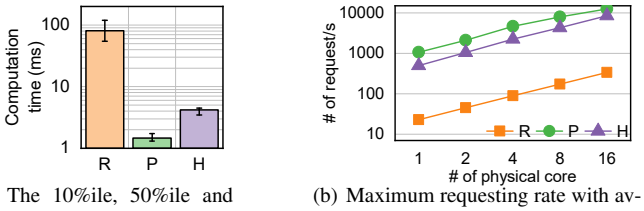
B. Motivation

On the way of improving the performance of ABR algorithms, the complexity of algorithms also goes up. Specifically, sophisticated algorithms are difficult to be practically deployed on the client or server side due to the following reasons.

Client-side implementation. As the conventional implementation solution for existing ABR algorithms [5, 6], implementing the heavyweight ABR models on resource-limited clients confronts a series of challenges. First, for neural network-based computational models [3, 9], when users watch videos from web browsers, loading heavyweight computation models will drastically increase the page load time by several seconds (§VI-B), which might make impatient users leave the page [22]. Second, for optimization-based models, solving complex optimizations on end devices with constrained computation resources will introduce excessive delay by up to 1000x compared to heuristic-based baselines (§VI-B). For example, the long decision latency of RobustMPC [8] sometimes might even exceed the video chunk lengths (often 2 to 4 seconds [11]) and severely degrade the QoE of videos. Moreover, with the recent development of live streaming [21], shorter chunks (and even frame-level decisions) will exacerbate the problem. Finally, additional software plugins (*e.g.*, TensorFlow [23]) might be required on video clients, which further poses barriers for large-scale deployments [24]. Therefore, the deployment practicality of ABR algorithms severely hinders the exploration of better ABR algorithms.

Therefore, to deploy sophisticated models on the client side, operators have to compromise performance to reduce overhead. RobustMPC [8] 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 could lead to a performance drop of up to 30%, which is worse than many strawman baselines [8]. 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 [8, 25].

Server-side implementation. Due to the complexity of recent sophisticated ABR algorithms, many recent solutions [3, 4, 8, 9, 24] offload the heavyweight online computation tasks to remote ABR servers. Although the server-side implementations could bring benefits of efficient updates of algorithms, this will significantly increase the operating expenses for content providers and thus are not scalable to large-scale deployment. As shown in Figure 1, due to high computation complexity (Figure 1(a)), the capacity of ABR servers



(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 [8], Pensieve [3], HotDASH [9]}. We build ABR servers with `tornado` [26] and test the capacity with `vegeta` [27] from another directly-connected server.

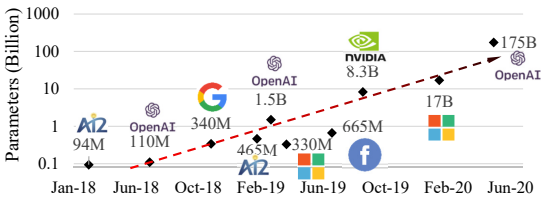


Figure 2. Parameter counts of several recent pretrained language models are going up sharply. Part of this figure was adapted from [29].

ranges from 20-1000 requests/second/core (Figure 1(b)) even accelerated with GPUs. However, there might be up to millions of concurrent connections for even one streaming video [28]. Moreover, according to our measurements in Kuaishou, one of the largest short video providers in China, there are 20 billion videos provided by Kuaishou on the Internet. Thus, introducing remote ABR servers may increase the OPEX by up to millions of dollars (§VI-C) for content providers, which makes the deployment of those algorithms impractical in large-scale real-world scenarios. Moreover, the round-trip latency between the clients and ABR servers may also be intolerable for delay-sensitive scenarios such as live streaming [21].

As discussed above, both methods fail to achieve high performance and scalability at the same time. In this way, algorithm designers would be constrained in considering the practicality of ABR algorithms with deployable methods, which will limit the potential capability of ABR. Moreover, to seek better performance, a trend of designing more and more heavyweight models has been observed in other communities (e.g., neural language processing in Figure 2). A similar trend starts to evolve in the scenarios of ABR algorithms: from 5 layers in 2017 [3], to the sophisticated combination of tens of fully connected, convolution, and recurrent layers in 2019 [17]. Employing more sophisticated algorithms in future ABR designs will make the matter worse [21].

In response, to deploy heavyweight ABR algorithms online, PiTree introduces a teacher-student learning-based method and converts ABR algorithms into decision trees. Our observation is that although ABR algorithms are more and more heavyweight in the design phase, the teacher-student learning enables us to convert the finetuned models into lightweight ones for online deployment.

III. PiTree Design

In this section, we articulate our design choice of adopting decision trees and design challenges (§III-A), and then introduce our decision tree conversion algorithm (§III-B).

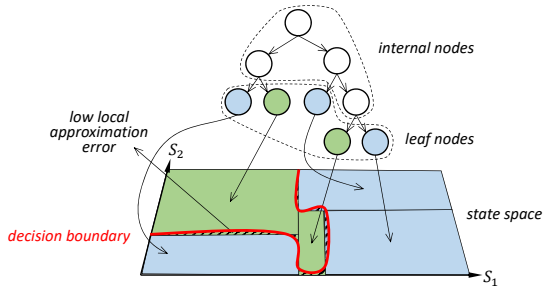


Figure 3. The decision tree can approximate the original decision boundary.

A. Design Choices and Challenges

As introduced in §I, 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 [18], nonlinear regression [19], and policy sketches [30], *etc*.

Adopting decision trees. PiTree employs decision trees due to the following reasons.

- The rich expressiveness of decision trees enables the high faithfulness of conversion because they are non-parametric and could represent complex policies [31]. As illustrated in Figure 3, decision trees can efficiently approximate the original algorithm even with highly nonlinear decision boundaries 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 $<1\%$ (§VI-B).
- 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).

Therefore, we adopt decision trees as our conversion target for PiTree. Our empirical results in §VI-A and VI-D demonstrate that the decision tree (in the simplest way) could maintain the performance degradation within 3% for state-of-the-art ABR algorithms.

Design challenges. Since decision tree training is a supervised learning method, it is designed to optimize the loss function (usually the average prediction accuracy [32]) with a labeled dataset under the distribution of the whole state space:

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

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 get the probability distribution of all the state space 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 [4, 8], which is both inefficient and may be biased from real-world scenarios. The dimension of the state space is often high (25 dimensions in Pensieve [3]), 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* [3, 6] and simulate ABR algorithms with real-world network traces. Compared to packet-level emulations, virtual players are fast and efficient since they only calculate chunk-level information. We then collect the state-action pairs during simulations and train decision trees with those pairs.

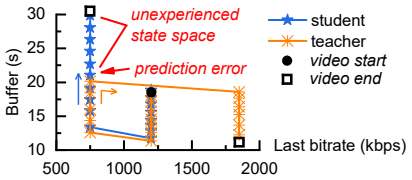


Figure 4. Without teacher-student learning, one wrong prediction may drive the student off teacher's trajectory in the state space.

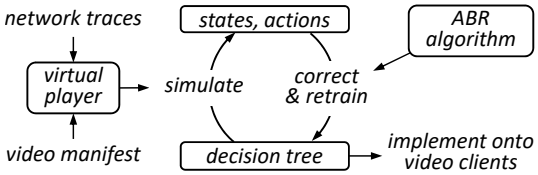


Figure 5. PiTree Overview.

Moreover, 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, the performance of the converted decision tree might still be degraded. As shown in Figure 4, 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 prior knowledge about that region of state space. Those mistakes will further drive the decision tree off the trajectory and worsen the performance. In response, PiTree continuously simulates the decision tree and lets the original ABR algorithm (teacher) correct the decisions made by that decision tree (student). The decision tree will gradually learn how to make decisions in the whole state space.

B. Decision Tree Generation

The overview of PiTree is presented in Figure 5. To convert sophisticated ABR algorithms into decision trees, PiTree uses a *virtual player* [3, 4, 6] to simulate the dynamics of a real video player efficiently and employs teacher-student learning [20, 33] to improve the faithfulness of the decision tree. With teacher-student 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. Similar to the training procedure of existing deep learning-based ABR algorithms [3, 9], the

conversion procedure is fully conducted within the virtual player, and does not require online implementation.

Specifically, as shown in Algorithm 1, the decision tree training algorithm contains the following steps:

Step 1: Initialization. To cope with dataset collection challenge as introduced in §III-A, for each ABR algorithm π^*, PiTree first simulates the algorithm in a virtual player to collect initial state-action pairs $(\mathbb{S}, \mathbb{A})$ (line 1). The virtual player is a trace-based chunk-level simulator to mimic the behaviors of an actual video player with traces and video manifests and has been widely used in training [3, 6, 9]. In reality, content providers could use public network traces [34, 35] or collected historical traces [4] for simulation. 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 teacher-student 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) [32], a well-adopted decision tree training algorithm (line 3). The decision tree takes 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:

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

where $r = \pi(s), r^* = \pi^*(s)$. 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*. The squared-loss function also enables better theoretical analysis (§IV). The CART algorithm then greedily splits the samples into leaf nodes to minimize the loss function until either (*i*) the number of leaf nodes of the decision tree reaches the maximum threshold set by network operators, or (*ii*) all samples have been completely split.

Step 3: VirtualPlay. As we discussed in §III-A, due to the cascading effects, the decision trees optimized from ***Step 2*** may still perform badly with new traces. 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 faced with the states fed in training, independently simulating π_i might lead to poor performance. Many states in $\mathbb{S}_i$ in the simulation might not be experienced by student π_i during the training in this iteration. Therefore, we still need to correct the decision tree policy in the following step.

Step 4: Correction. To correct the actions collected from ***Step 3***, 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.

Algorithm 1: PiTree training procedure with teacher-student learning.

Input: ABR Algorithm π^*.
Output: Decision Tree π_M.
1 $(\mathbb{S}, \mathbb{A}) \leftarrow$VirtualPlay($\pi^*$)
2 $i \leftarrow 0$
3 **foreach** $i \in [1, \cdots, M]$ **do**
4 $\quad \pi_i \leftarrow$TrainDT($\mathbb{S}, \mathbb{A}$)
5 $\quad (\mathbb{S}_i, \mathbb{A}_i) \leftarrow$VirtualPlay($\pi_i$)
6 $\quad \mathbb{A}_i^* \leftarrow$Predict($\pi^*, \mathbb{S}_i$)
7 $\quad$ Aggregate $\mathbb{S} \leftarrow \mathbb{S} \cup \mathbb{S}_i, \mathbb{A} \leftarrow \mathbb{A} \cup \mathbb{A}_i^*$

As introduced above, there are two hyper-parameters set by the network operators: the number of leaf node N, which represents the expressive ability of the decision tree, and the maximum iteration number M, which controls the training procedure. For the setting of N, usually a more complex ABR algorithm will needs a higher N. Our empirical results show that N could also be determined by observing the training loss. For the setting of M, network operators can determine M by observing the training curve, as shown in Figure 13. Operators can also refer to the termination in neural network training, e.g., automatically terminating the conversion when the accuracy no longer changes [36]. We analyze our parameter settings on M and N and their sensitivity with several ABR algorithms in detail in §VI-C. Moreover, our evaluation in §VI-C 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.

IV. Theoretical Analysis

In this section, we provide the theoretical analysis of the performance of PiTree. To help operators to better understand the behaviors of the converted decision trees, we answer two questions on the worst-case performance of PiTree:

- **Prediction error.** For *any* deterministic ABR algorithms emulated in *any* traffic traces, is there an upper bound for the bitrate prediction error of the decision tree? (§IV-A)
- **Generalization loss.** During online deployment, is there a bound for the generalization loss if the online scenarios are different from emulations? (§IV-B)

A. Prediction Error

We first show that the prediction error of the converted decision tree compared to the original ABR algorithm is bounded. We define the root-mean-square error (RMSE) of PiTree against the original algorithm during conversion as:

$$RMSE_T := \sqrt{\frac{1}{T}\sum_{t=1}^{T}\left(\hat{r}^{(t)} - r^{*(t)}\right)^2} \qquad (3)$$

where $\hat{r}^{(t)}$ and $r^{*(t)}$ are the bitrate decisions made by $\hat{\pi}$ and π^* at time t. We then have the following theorem to bound the RMSE of PiTree:

Theorem 1. *Among all policies* Π *generated with PiTree, for a* T*-chunk video adopted in the virtual player, there exists one policy* $\hat{\pi} \in \Pi$ *that satisfies:*

$$RMSE_T \leqslant \sqrt{\frac{1+\log T}{T}}(R_{max} - R_{min}) \tag{4}$$

Proof of Theorem 1. See Appendix. □

$\hat{\pi}$ 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. Theorem 1 provides an upper bound of the bitrate prediction error for network operators. As presented in the Equation 4, the upper bound decreases with the increase of the number of video chunks T. This is due to the cascading effect as shown in Figure 4: A longer trajectory will amplify the difference between the original ABR algorithm and the decision tree during training. Therefore, the converted decision tree will receive better corrections from the teacher during conversion, and become more robust after conversion.

With the theoretical guarantee on the worst-case performance, network operators would be more confident in deploying PiTree in critical scenarios. Note that Theorem 1 guarantees the *worst-case* performance degradation of the algorithm, which is much larger than the actual values with real-world traces. Our empirical results demonstrate that the average performance degradation on the QoE is less than 3% over different types of traces (§VI-E).

B. Generalization Loss

With the guarantees on the conversion error during emulation, network operators may also be interested in the generalization performance of PiTree when the online deployment scenarios are different from the emulation. We thus have the following upper bound of the average optimization loss $\ell\left(\hat{\pi}(s); \pi^*(s)\right)$ when the decision tree generated by PiTree is independently deployed online:

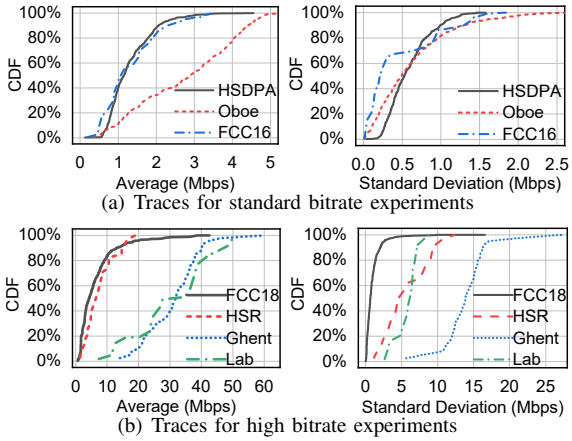
Theorem 2. *For any* $\delta > 0$*, with training loss* ε_M*, there exists a policy* $\hat{\pi} \in \{\pi_1, \cdots, \pi_M\}$ *s.t. the average optimization loss satisfies:*

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

with probability at least $1-\delta$ *as long as* $M = O(T\log(1/\delta))$*.* M *is the number of iterations in training.* T *is the number of chunks in the video.*

Proof of Theorem 2. See Appendix. □

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 (expressive ability of decision tree), and could be bounded by Theorem 1. We also empirically evaluate the generalization performance of PiTree when the statistical characteristics of the training traces are different from the online test traces, resulting in negligible generalization loss (§VI-C). Further evaluation of PiTree with a series of real-world experiments demonstrates the strong generalization ability of PiTree in the real world (§VI-D).



(a) Traces for standard bitrate experiments

(b) Traces for high bitrate experiments

Figure 6. The statistics of the throughput of network traces.

V. Implementation

Due to the fast increase of Internet bandwidth (tens of Mbps on average [37]), existing videos and network traces for evaluation [3, 4, 9] will behave trivially on bitrate decisions around 1 Mbps. Therefore, we collect several new traces together with 4G network traces measured by us (§V-A) and compile new videos at ultra-high bitrates (§V-B). Traces and videos have been released at https://transys.io/pitree-dataset/. We also introduce experiment setups of PiTree in §V-C.

A. Network Traces

As for network traces, to make a fair comparison, we adopt the traces used in the evaluation of previous work [3, 4, 8] to evaluate PiTree. Network traces include measurements from Norway's 3G HSDPA [35] (denoted as **HSDPA**), US FCC broadband measurement results in 2016 [34] (denoted as **FCC16**) and traces provided by **Oboe** [4]. These three sets of traces are statistically different, as presented in Figure 6(a).

Meanwhile, with the development of the networking infrastructure in recent years, the bandwidth of traces evaluated in existing papers (up to several Mbps) is insufficient for the Internet in 2020 (tens of Mbps). Therefore, as illustrated in Figure 6(b), we adopt four recent sets of network traces reflecting different variations:

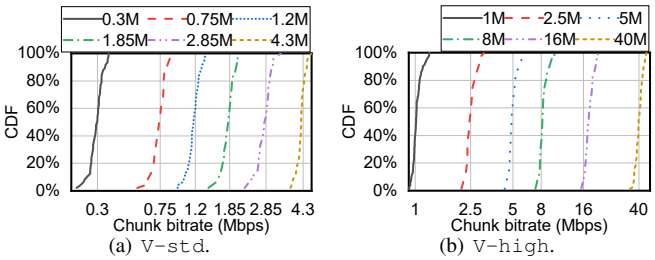
- **FCC18** is the measurement results of the broadband network in 2018 provided by FCC [34], with a median bandwidth improvement by 4.04× compared to 2016.
- **HSR**. We adopt the 4G measurements on high-speed rails in 2018 to construct a scenario with violently fluctuating 4G bandwidths [37].
- **Ghent**. We use the 4G measurements on foot, bicycle, bus, tram, train, and car in 2016 by Ghent University, which are moderately fluctuating 4G bandwidths [38].
- **Lab**. Finally, we also measure the indoor 4G bandwidth. We interchangeably use four congestion control algorithms (BBR, Cubic, Vegas, and HighSpeed). The indoor traces construct a scenario with gently fluctuating 4G bandwidths.

B. Video Sample

We evaluate PiTree with two video samples and present the characteristics in Table II. To make a fair comparison, we

Table II
CHARACTERISTICS OF VIDEO SAMPLES.

Video	Bitrates (Mbps)	Duration	Chunk length
V-std	0.3, 0.75, 1.2, 1.85, 2.85, 4.3	193s	4s
V-high	1, 2.5, 5, 8, 16, 40	320s	4s



(a) V-std. (b) V-high.

Figure 7. The distribution of bitrates of two video samples.

employ the "EnvivoDash3" video (denoted as V-std), which has been used in prior work [3, 4]. However, the highest bitrate of the V-std video is 4.3Mbps, which is much lower than the network bandwidth nowadays, as shown in Figure 6(b). In this case, most decisions of ABR algorithms would be trivial by always selecting the highest bitrate.

Therefore, we refer to the bandwidth levels of YouTube [39] and construct a new video sample with the highest bitrate of 40Mbps based on MPEG-DASH. Since video samples are encoded with variable bitrate (VBR), the actual bitrate of each chunk fluctuates around the targeted bitrate. We present the distribution of the bitrate of all chunks of our V-high video together with the V-std video in Figure 7.

C. Experiment Setup

ABR algorithms. We apply PiTree over the following state-of-the-art ABR algorithms based on the source codes provided by the authors:

- RobustMPC [8] employs integer programming to optimize bitrates with buffer occupancy and network throughput.
- Pensieve [3] models the bitrate selection process with Reinforcement Learning (RL) and makes predictions based on 25 states with a neural network.
- HotDASH [9] extends Pensieve and uses two cascaded neural networks to make ABR decisions.

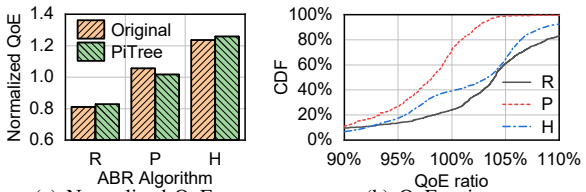
QoE metrics. To make a fair comparison, we employ the QoE metrics adopted in the ABR algorithms above. The QoE metric can be expressed as: $QoE =$

$$\frac{1}{T}\left(\sum_{t=1}^{T} q(R_t) - \mu \sum_{t=1}^{T} \beta_t - \sum_{t=1}^{T-1} |q(R_{t+1}) - q(R_t)|\right) \quad (6)$$

where R_t represents the bitrate of chunk t. T is the total number of chunks. β_t is the rebuffering time of chunk t. $q(\cdot)$ is the utilization function as defined in Table III. To better illustrate the individual performance of different parts of QoE,

Table III
QOE METRICS CONSIDERED IN OUR EVALUATION [3, 7].

QoE_{lin}	$q(R) = R$	$\mu = R_{max}$
QoE_{log}	$q(R) = \log(R/R_{min})$	$\mu = \log(R_{max}/R_{min})$
QoE_{hd}	$q(R) = \{1, 2, 3, 12, 15, 20\}$	$\mu = 8$



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

Figure 8. Overall average results of PiTree. {R, P, H} refer to {RobustMPC, Pensieve, HotDASH} respectively.

we consider three choices of $q(R_t)$ in prior work [3, 5, 8]. Three terms in Equation 6 respectively refer to video quality, rebuffer penalty and smoothness penalty.

Browser implementations. We use the virtual player in [3] for the decision tree training. We convert the decision tree into JavaScript codes with `sklearn-porter`[1]. We migrate the decision tree generated by PiTree into `dash.js` [10] and compress JavaScript codes with the `UglifyJS` plugin in the `Grunt.js`[2]. We use Mahimahi [40] to emulate the network conditions and set the round-trip time (RTT) to 80ms, which are the same with Pensieve [3].

Parameter settings. As we discussed in §III-B, the number of leaf nodes should be adjusted for different ABR algorithms. According to the complexity of ABR algorithms, we set the number of leaf nodes to 500, 100, and 100 for RobustMPC, Pensieve, and HotDASH. We discuss our settings in §VI-C.

VI. EVALUATION

We apply PiTree over three state-of-the-art ABR algorithms [3, 8, 9], with seven sets of network traces, two types of videos, and on three QoE metrics. We evaluate PiTree in the following aspects:

- ***QoE Maintenance.*** Our evaluation results demonstrate that the average performance degradation caused by PiTree is within 3% for all three ABR algorithms (§VI-A).
- ***Overhead.*** 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 (§VI-B).
- ***Deployment Efforts.*** Our evaluation shows that PiTree could save considerable operating expenses, consume acceptable additional offline training time, and have robust parameter settings and strong generalization ability (§VI-C).
- ***Real-world Performance.*** We evaluate PiTree in the wild with four wide area networks. Experiments demonstrate that PiTree could maintain the performance in the real world even trained with other traces (§VI-D).
- ***PiTree Deep Dive.*** We finally demonstrate that PiTree improves the bitrate prediction error on three ABR algorithms and outperforms other conversion options. Evaluation also validates the theoretical bounds of PiTree (§VI-E).

A. QoE Maintenance

We demonstrate the performance maintenance of PiTree by comparing the QoE of original algorithms and decision trees

[1]https://github.com/nok/sklearn-porter
[2]https://github.com/gruntjs/grunt-contrib-uglify

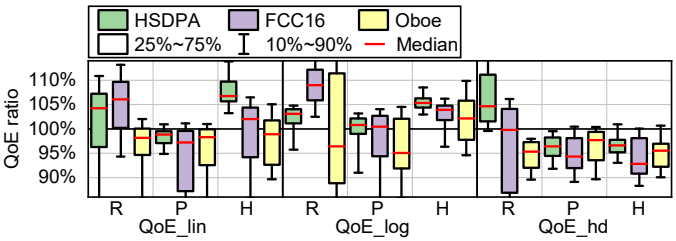


Figure 9. QoE ratio of PiTree on different ABR algorithms and QoE metrics (measured with the `V-std` video and standard bitrate traces).

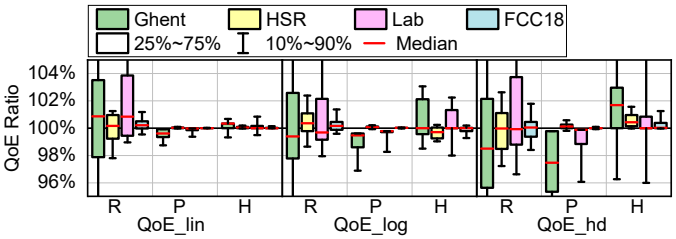


Figure 10. QoE ratio of PiTree on different ABR algorithms and QoE metrics (measured with the `V-high` video and high bitrate traces).

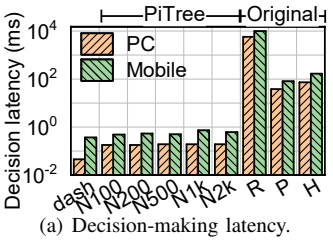
converted with PiTree. We thus measure the ratio of QoE by the PiTree-generated decision trees and the original algorithms. A QoE ratio of less than 100% indicates a performance degradation. Since the QoE spans across positive and negative values, we normalize all the QoE values into a distribution with mean value as 1 and standard deviation as 1. We first measure the average normalized QoE and average QoE ratio across three types of QoE metrics and all traces, as shown in Figure 8. The average performance degradation is less than 3% for three algorithms (average QoE ratio of Pensieve is 97% in Figure 8(b)), which is negligible compared to the performance improvement achieved by new algorithms.

We further present the detailed results on different traces and different QoE metrics in Figures 9 and 10. Among them, Figure 9 reports the experiment results with `V-std` and respective standard bitrate traces (Figure 6(a)) and Figure 10 summarizes the `V-high` video and high bitrate traces (Figure 6(b)). The 10th, 25th, 50th, 75th, and 90th percentiles of QoE ratios on three QoE metrics, seven sets of traces, and three ABR algorithms are presented. 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.

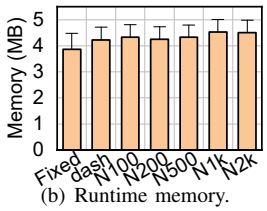
B. Overhead

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.

Page size. We first measure the HTML page size and present the results in Figure 12(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` [23]. The page load time is

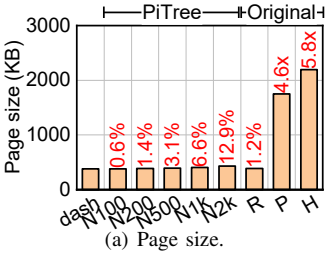


(a) Decision-making latency.



(b) Runtime memory.

Figure 11. Overhead of PiTree. `N100`: the decision tree with 100 leaf nodes. The error bar of runtime memory represents the peak value.



(a) Page size.

(b) Operating expenses.

Figure 12. Page size and operating expenses of PiTree.

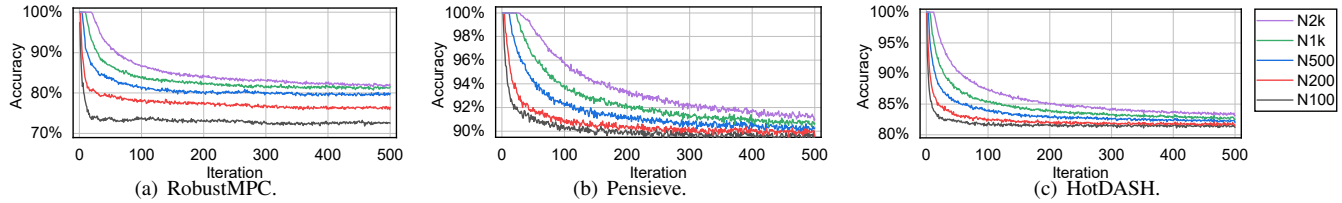
thus increased by 10 seconds when the goodput is 1200kbps. Users have to wait for a long time before the video can play, which might drive some of them to leave the page [22]. In contrast, results show that even with 2000 leaf nodes (`N2k`), the page size of PiTree is only increased by 13%. Moreover, our experiments in §VI-A 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%. The negligible page size difference only introduces an additional page load time by 10ms.

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 11(a), the decision latency of original algorithms is found to be 1s, 3-4 magnitudes larger than that of PiTree-based algorithms. Such a long decision latency will not only impair the QoE due to the out-of-date information [8] but also stall the video player when the video chunk length is less than the average decision latency (*e.g.*, 2s in [11]). In contrast, the average decision-making latency of PiTree is significantly reduced to less than 1ms, which is at the same magnitude as the default ABR algorithm in `dash.js`.

Memory utilization. We finally measure the average runtime JavaScript heap memory with the `memory` API in Chrome DevTools [41]. We implement a *fixed bitrate algorithm* as a baseline, which constantly selects the lowest bitrate, to eliminate the influence of other functions in the video player. As shown in Figure 11(b), 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.

C. Deployment Efforts

We evaluate the operating expenses and offline conversion time during the deployment of PiTree. Moreover, we also



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

Figure 13. Single prediction accuracy of PiTree on the training set at different iterations. The accuracy goes down because $(\mathbb{S}, \mathbb{A})$ lacks samples in the first several iterations and decision trees are overfitted.

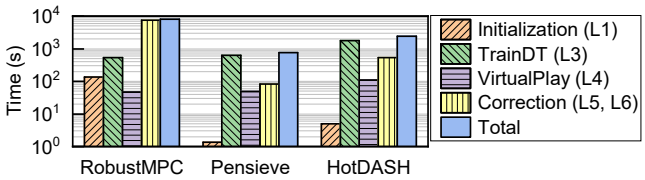


Figure 14. Offline training time breakdown.

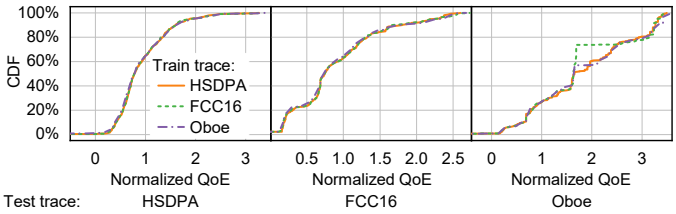


Figure 15. Normalized QoE_{lin} of PiTree-over-Pensieve when test traces are statistically different from training traces.

analyze the sensitivity and generalization ability of PiTree in different scenarios, indicating that network operators do not need to pay many extra efforts to finetune the parameters.

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]. As shown in Figure 12(b), for large content providers such as YouTube, with more than one billion hours of video clips being watched daily[4], 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.

Offline conversion cost. We break down the training time for each algorithm for 500 iterations and present the results in Figure 14 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 PiTree needs running offline only once.

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. The results are presented in Figure 13. For RobustMPC, as shown in Figure 13(a), the accuracy of decision trees 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 13(a)). Decision trees with different numbers of leaf nodes for Pensieve and HotDASH (Figure 13(b), 13(c)) demonstrate a similar relationship. The converged performance of PiTree is hardly affected as long as the number of leaf nodes is above a certain level, indicating strong robustness towards the number of leaf nodes.

Generalization of PiTree. When deployed in practice, the statistical distributions of network traces in test might be different from those in training. Therefore, we want to measure the generalization ability of PiTree when network traces change *without retraining* the decision tree. We measure the normalized QoE_{lin} of the decision trees trained with different traces in different test traces. The decision trees evaluated here are set to imitate the behaviors of Pensieve. As shown in Figure 15, even if decision trees are trained with different traces, they perform similar 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 15 is higher than 97%. Performance over Oboe on some certain traces is degraded a little since the traffic distribution of Oboe is quite different from those of the other two sets of traces (Figure 6). Experiment results for other ABR algorithms and QoE metrics are similar (not presented), which demonstrate the strong generalization ability of PiTree. Moreover, online parameter tuning [4] could be employed to further enhance the generalization ability. We leave the large-scale deployments as our future work.

D. Real-World Experiments

We evaluate PiTree over RobustMPC in the wild using three wide area networks (two broadband and one cellular): Beijing-Shenyang (BJ-SY) with broadband access, Beijing-Hangzhou (BJ-HZ) with broadband access, and a 4G cellular network of China Mobile (CM-4G). In these experiments, a client, running

[3]https://aws.amazon.com/ec2/pricing/reserved-instances/pricing/
[4]http://www.businessofapps.com/data/youtube-statistics/

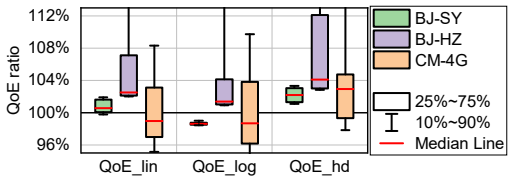


Figure 16. QoE ratio of PiTree on RobustMPC in the real-world experiments.

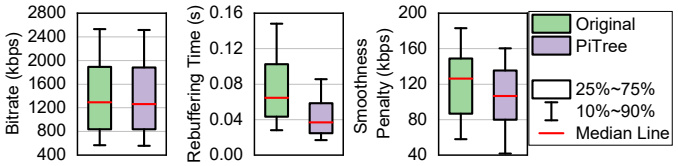
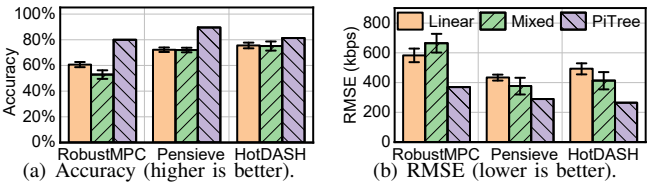


Figure 17. Raw metrics of Pensieve and PiTree-over-Pensieve.



(a) Accuracy (higher is better). (b) RMSE (lower is better).

Figure 18. The training accuracy and RMSE of PiTree against two baselines. Error bar represents the standard deviation.

on a Windows 10 laptop and Firefox 73 browser, contacted the video server as described in §V-C located in Beijing. On each network, we loaded our test video 100 times with each scheme, randomly selecting the order among them. Since the network bandwidth in the real world is much higher than 4.3Mbps, we use the `V-high` video for evaluation.

We test the video for more than 50 hours of view time in total in February 2020. We present the experiment results in Figure 16. As shown in the figure, PiTree could maintain the QoE ratio of the original ABR algorithm in most cases, with the minimum 50th percentile of 98%. Note that the PiTree decision trees evaluated here were solely trained using the high bandwidth traces presented in Figure 6(b). Nonetheless, even on these new networks, the decision trees PiTree could also behave faithfully in the bitrate decisions. Experiments with other ABR algorithms show similar results.

E. PiTree Deep Dive

Finally, we deep dive into PiTree in three directions: First, we present the difference of raw metrics during the conversion. Second, we want to know how the decision tree behave compared to other conversion options and whether our design choice in §III-A holds. Finally, we demonstrate that the theoretical bounds in §IV-A hold in our experiments.

Raw metrics decoupling. Besides the QoE ratio evaluated in §VI-A, we measure the raw metric of different traces. According to the QoE definition in Equation 6, raw metrics include bitrate, rebuffering time, and smoothness penalty. We present different percentiles of three metrics of the original Pensieve model and the converted model from PiTree in Figure 17. On one hand, as the major contributing part to the QoE, PiTree introduces a performance degradation of 2.6% on median compared to the original model. On the other hand, the rebuffering time and smoothness penalty of PiTree have been slightly improved. Evaluation with RobustMPC demonstrates similar results. We hypothesize such phenomenon to be an observation of the *lottery ticket hypothesis* in the machine learning community [42]: Sometimes the performance of the pruned model is reliably better than the original model due to the reduction of the overcomplexity of the original model and the improvement of convergence efficiency. We leave the further exploration on the model overcomplexity of ABR algorithms as our future work.

Model conversion comparison. We measure the accuracy and root-mean-square error (RMSE) of the decisions made by PiTree compared to the original ABR decisions. As baselines, we also implement two regression-based methods introduced in recent interpretation methods: (*i*) linear regression [18] and (*ii*) mixed regression with expectation-maximization (EM) [19]. As both methods are designed for local predictions, to make a fair comparison, we run the baselines in the following way: at training, we first use k-means clustering to cluster the samples into k groups. We then train the *linear* and *mixed* method inside each group. When predicting a new sample, we first find the nearest group to the new sample and apply the results of that group. We vary k from 1 to 50 and report the best results for the two baselines. To eliminate the randomness caused by the split of training and validation data, we repeat the experiments for 100 times. Results of the `V-std` video and standard bitrate traces are shown in Figure 18. The experiments with the high bitrate video and traces show the similar results. PiTree outperforms the other two baselines (linear regression and mixed regression) on both accuracy and RMSE, which indicates that our design choice in §III-A of selecting the decision tree as our conversion target is reasonable.

Theoretical bound analysis. From the RMSE results in Figure 18(b), we can also verify the theoretical bound derived in §IV-A. As for the standard bitrate experiments, we have R_{max} =4300kbps, R_{min} =300kbps, and $T = 49$. Thus the upper bound of RMSE normalized by the bitrate is:

$$\frac{RMSE}{R_{max}-R_{min}} \leqslant \sqrt{\frac{1+\log T}{T}} = 0.32$$

On the other hand, three RMSE values of PiTree in Figure 18(b) are less than 0.1 after normalization. Therefore, all the results of our experiments under different conditions satisfy the upper bound above. Note that the RMSE bound is much higher than the average RMSE of PiTree in this case since the theoretical bound guarantees the *worst-case* performance to deliver confidence to network operators. Empirical results often demonstrate much better performance than the theoretical bound. The RMSE bound would be tighter with longer training videos as the right-hand-side term decreases with T.

VII. Discussions

In this section, we discuss and highlight several potential directions as our future work.

Insights behind PiTree. As evaluated in §VI-A, converted decision trees exhibit comparable performance to original ABR models, including ILP and neural networks. Our main observation from the evaluation result is that in the sequential

decision-making scenarios such as ABR streaming, the main reason of employing sophisticated models is not the policy itself are that complex, but those sophisticated models are easy to train. For example, Pensieve trains ABR model with deep reinforcement learning [3], where neural networks can be updated with many mature algorithms (e.g., policy gradients [43]). However, directly training decision trees from scratch is difficult to achieve the same performance: without labeled datasets in the sequential decision-making process, decision trees are not easy to be efficiently updated during training due to its non-parametricity. PiTree overcomes this problem by continuously collecting trajectories from the teacher and train the decision tree with labeled samples.

Dynamicity of PiTree. Due to the complexity of the decision process of state-of-the-art algorithms (e.g., neural network), the parameters inside are difficult to be dynamically adjusted. Converting complicated ABR algorithms also enables us to dynamically adjust the structure of the decision tree to adapt to different scenarios. For example, with the help of decision tree adaption methods [44], network operators could design ABR algorithms that are adaptive to different network scenarios. Network operators could also cooperate with the parameters in PiTree with recent online parameter tuning mechanisms for better adaption in different scenarios [4]. We leave the analysis of decision tree dynamics as our future work.

Fairness of learning-based ABR. The fairness of ABR algorithms is another critical issue and has been widely discussed in the previous work [11]. Network operators should avoid providing unfair services to different users. However, some recent clues indicate that DNNs are likely to trade a little fairness for a more efficient overall performance [45]. Moreover, since the policies are nontransparent to network operators, it is unknown whether the adoption of DL-based networking systems will impair the fairness among users. Converting DNN-based policies to decision trees might address the fairness problem to some extent by transparentizing the decision policies. Nonetheless, further research with advanced methods (e.g., fair decision trees [46]) is still required to ensure fairness of learning-based ABR algorithms.

VIII. Related Work

There is little prior work on how to deploy heavyweight ABR algorithms into client-side video players. In terms of approaches and motivations, besides recent research efforts on optimizing the QoE as introduced in §II-A, there are also several lines of work that are related to PiTree:

Decision trees in video streaming. Besides PiTree, decision trees have also been applied to many subfields in video streaming, most of which are used for prediction. For example, Hammed et al. employed decision trees to predict perceptual video quality from compressed bitstream [47]. Balachandram et al. adopted decision trees to predict user engagement based on video session quality [48]. All efforts above are orthogonal to PiTree and could be integrated together for better performance. However, few research efforts directly use a decision tree for bitrate adaption. The main reason is that decision trees are mainly suitable for *supervised learning* scenarios (e.g., prediction) with independent actions. Due to the cascading effect of ABR, directly training the decision tree is difficult to achieve the same performance [3]. PiTree overcomes this issue by letting the original ABR algorithm teach the student decision tree with its samples.

Complex model deployment. In other communities, there are also some recent work on how to deploy sophisticated models in practice. One set of solutions introduce new acceleration devices (e.g., DSP [49], GPU [7], or ASIC [50]). Another way to deploy complex models (especially DNNs) is to prune and compress neural networks by removing insignificant filters [51, 52]. 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 as long as they are non-stochastic. Moreover, instead of introducing new expenses [7, 49], the online overhead and deployment efforts of PiTree are negligible (§VI-B, §VI-C).

Teacher-student learning for sequential decisions. Using demonstrations from teachers has a series of successful use cases in the sequential decision-making process, such as robotics control [33]. Researchers from the machine learning community have also proposed different enhancement to the basic procedure of teacher-student learning, including innovations in the resampling of dataset [20], designs of teachers' and students' structures [53], and improving the teachers' exploration efficiency [54, 55]. PiTree employs the existing advances in teacher-student learning and analyzes the theoretical performance in the scenario of ABR streaming. Advanced teacher-student learning algorithms might also improve the performance, which is left as our future work.

Other practical issues of learning-based ABR. Besides the heavyweightness of complex ABR models, there are also several issues that need to be addressed before the large-scale deployment of learning-based ABR algorithms. Some recent papers focus on the interpretability [56, 57] and verification [58] of learning-based ABR algorithms. By converting complex models into self-interpretable decision trees, PiTree could offer better interpretability and transparency for network operators compared to original ones. Other researchers focus on speeding up the training procedure and eliminating the influence of network traces by proposing novel training methodology [59, 60], which could also be adopted in PiTree to further improve the training performance.

IX. Conclusions

In this paper, we propose PiTree, a new framework to generally make the deployment of sophisticated ABR algorithms practical in the real world. PiTree faithfully converts different ABR algorithms into decision trees with the help of offline teacher-student 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.

REFERENCES

[1] Z. Meng, J. Chen, Y. Guo, C. Sun, H. Hu, and M. Xu, "Pitree: Practical implementation of abr algorithms using decision trees," in *Proc. ACM Multimedia*, 2019, pp. 2431–2439.

[2] "Mobile accounted for 62 percent of online video views," https://www.statista.com/statistics/444318/mobile-device-video-views-share/.

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

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

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

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

[7] H. Yeo, Y. Jung, J. Kim, J. Shin, and D. Han, "Neural adaptive content-aware internet video delivery," in *Proc. USENIX OSDI*, 2018, pp. 645–661.

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

[9] S. Sengupta, N. Ganguly, S. Chakraborty, and P. De, "Hotdash: Hotspot aware adaptive video streaming using deep reinforcement learning," in *Proc. IEEE ICNP*, 2018, pp. 165–175.

[10] T. Stockhammer, "Dynamic adaptive streaming over http – standards and design principles," in *Proc. ACM MMSys*, 2011, pp. 133–144.

[11] J. Jiang, V. Sekar, and H. Zhang, "Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive," in *Proc. ACM CoNEXT*, 2012, pp. 97–108.

[12] 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 J. Sel. Areas Commun.*, pp. 719–733, 2014.

[13] C. Wang, A. Rizk, and M. Zink, "Squad: A spectrum-based quality adaptation for dynamic adaptive streaming over http," in *Proc. ACM MMSys*, 2016, pp. 1–12.

[14] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson, "A buffer-based approach to rate adaptation: Evidence from a large video streaming service," in *Proc. ACM SIGCOMM*, 2014, pp. 187–198.

[15] L. De Cicco, V. Caldaralo, V. Palmisano, and S. Mascolo, "Elastic: A client-side controller for dynamic adaptive streaming over http (dash)," in *Proc. IEEE International Packet Video Workshop*, 2013, pp. 1–8.

[16] B. Wang and F. Ren, "Towards forward-looking online bitrate adaptation for dash," in *Proc. ACM Multimedia*, 2017, pp. 1122–1129.

[17] T. Huang, C. Zhou, R.-X. Zhang, C. Wu, X. Yao, and L. Sun, "Comyco: Quality-aware adaptive video streaming via imitation learning," in *Proc. ACM Multimedia*, 2019, pp. 429–437.

[18] M. T. Ribeiro, S. Singh, and C. Guestrin, "Why should i trust you?: Explaining the predictions of any classifier," in *Proc. ACM KDD*, 2016, pp. 1135–1144.

[19] W. Guo, D. Mu, J. Xu, P. Su, G. Wang, and X. Xing, "Lemna: Explaining deep learning based security applications," in *Proc. ACM CCS*, 2018, pp. 364–379.

[20] O. Bastani, Y. Pu, and A. Solar-Lezama, "Verifiable reinforcement learning via policy extraction," in *Proc. NeurIPS*, 2018, pp. 2494–2504.

[21] G. Yi, D. Yang, A. Bentaleb, W. Li, Y. Li, K. Zheng, J. Liu, W. T. Ooi, and Y. Cui, "The acm multimedia 2019 live video streaming grand challenge," in *Proc. ACM Multimedia*, 2019, pp. 2622–2626.

[22] S. S. Krishnan and R. K. Sitaraman, "Video stream quality impacts viewer behavior: Inferring causality using quasi-experimental designs," in *Proc. ACM IMC*, 2012, pp. 211–224.

[23] D. Smilkov, N. Thorat, Y. Assogba, C. Nicholson, N. Kreeger, P. Yu, S. Cai, E. Nielsen *et al.*, "Tensorflow.js: Machine learning for the web and beyond," in *Proc. SysML*, 2019, pp. 1–13.

[24] A. Ganjam, F. Siddiqui, J. Zhan, X. Liu, I. Stoica, J. Jiang, V. Sekar, and H. Zhang, "C3: Internet-scale control plane for video quality optimization," in *Proc. USENIX NSDI*, 2015, pp. 131–144.

[25] A. Beben, P. Wiśniewski, J. M. Batalla, and P. Krawiec, "Abma+: lightweight and efficient algorithm for http adaptive streaming," in *Proc. ACM MMSys*, 2016, pp. 1–11.

[26] "Tornado web server," https://www.tornadoweb.org/.

[27] "tsenart/vegeta: Http load testing tool and library. it's over 9000!" https://github.com/tsenart/vegeta.

[28] "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.

[29] C. Rosset, "Turing-nlg: A 17-billion-parameter language model by microsoft," https://www.microsoft.com/en-us/research/blog/turing-nlg-a-17-billion-parameter-language-model-by-microsoft/.

[30] A. Verma, V. Murali, R. Singh, P. Kohli, and S. Chaudhuri, "Programmatically interpretable reinforcement learning," in *Proc. ICML*, 2018, pp. 5045–5054.

[31] H. Blockeel and L. De Raedt, "Top-down induction of first-order logical decision trees," *Elsevier Artificial intelligence*, pp. 285–297, 1998.

[32] J. H. Friedman, R. A. Olshen, C. J. Stone *et al.*, "Classification and regression trees," *Belmont, CA: Wadsworth & Brooks*, 1984.

[33] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proc. AISTATS*, 2011, pp. 627–635.

[34] "Raw data - measuring broadband america," https://www.fcc.gov/reports-research/reports/measuring-broadband-america/raw-data-measuring-broadband-america.

[35] H. Riiser, P. Vigmostad, C. Griwodz, and P. Halvorsen, "Commute path bandwidth traces from 3g networks: Analysis and applications," in *Proc. ACM MMSys*, 2013, pp. 114–118.

[36] R. Caruana, S. Lawrence, and C. L. Giles, "Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping," in *Proc. NIPS*, 2001, pp. 402–408.

[37] L. Li, K. Xu, T. Li, K. Zheng, C. Peng, D. Wang, X. Wang, M. Shen, and R. Mijumbi, "A measurement study on multi-path tcp with multiple cellular carriers on high speed rails," in *Proc. ACM SIGCOMM*, 2018, pp. 161–175.

[38] J. Van Der Hooft, S. Petrangeli, T. Wauters, R. Huysegems, P. R. Alface, T. Bostoen, and F. De Turck, "Http/2-based adaptive streaming of hevc video over 4g/lte networks," *IEEE Commun. Letters*, pp. 2177–2180, 2016.

[39] "Recommended upload encoding settings - youtube help," https://support.google.com/youtube/answer/1722171.

[40] R. Netravali, A. Sivaraman, S. Das, A. Goyal, K. Winstein, J. Mickens, and H. Balakrishnan, "Mahimahi: Accurate record-and-replay for HTTP," in *Proc. USENIX ATC*, 2015, pp. 417–429.

[41] "Chrome devtools," https://developers.google.com/web/tools/chrome-devtools/.

[42] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," in *Proc. ICLR*, 2019.

[43] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Proc. NIPS*, 1999.

[44] J. Gama, R. Rocha, and P. Medas, "Accurate decision trees for mining high-speed data streams," in *Proc. ACM KDD*, 2003, pp. 523–528.

[45] M. Hardt, E. Price, N. Srebro *et al.*, "Equality of opportunity in supervised learning," in *Proc. NIPS*, 2016, pp. 3315–3323.

[46] S. Aghaei, M. J. Azizi, and P. Vayanos, "Learning optimal and fair decision trees for non-discriminative decision-making," in *Proc. AAAI*, 2019, pp. 1418–1426.

[47] A. Hameed, R. Dai, and B. Balas, "A decision-tree-based perceptual video quality prediction model and its application in fec for wireless multimedia communications," *IEEE Trans. Multimedia*, pp. 764–774, 2016.

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

[49] N. D. Lane, P. Georgiev, and L. Qendro, "Deepear: Robust smartphone audio sensing in unconstrained acoustic environments using deep learning," in *Proc. ACM Ubicomp*, 2015, pp. 283–294.

[50] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE Journal of Solid-State Circuits*, pp. 127–138, 2017.

[51] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," in *Proc. ICLR*, 2017, pp. 1–13.

[52] W. Chen, J. Wilson, S. Tyree, K. Weinberger, and Y. Chen, "Compressing neural networks with the hashing trick," in *Proc. ICML*, 2015, pp. 2285–2294.

[53] T. Matiisen, A. Oliver, T. Cohen, and J. Schulman, "Teacher-student curriculum learning," *IEEE Trans. Neural Netw. Learn. Sys.*, 2019.

[54] S. Levine and V. Koltun, "Guided policy search," in *Proc. ICML*, 2013.

[55] T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, D. Horgan, J. Quan, A. Sendonaris, I. Osband *et al.*, "Deep q-learning from demonstrations," in *Proc. AAAI*, 2018.

[56] A. Dethise, M. Canini, and S. Kandula, "Cracking open the black box: What observations can tell us about reinforcement learning agents," in *Proc. ACM NetAI*, 2019, pp. 29–36.
[57] Z. Meng, M. Wang, J. Bai, M. Xu, H. Mao, and H. Hu, "Interpreting deep learning-based networking systems," in *Proc. ACM SIGCOMM*, 2020, pp. 154–171.
[58] Y. Kazak, C. Barrett, G. Katz, and M. Schapira, "Verifying deep-rl-driven systems," in *Proc. ACM NetAI*, 2019, pp. 83–89.
[59] H. Mao, M. Schwarzkopf, S. B. Venkatakrishnan, Z. Meng, and M. Alizadeh, "Learning scheduling algorithms for data processing clusters," in *Proc. ACM SIGCOMM*, 2019, pp. 270–288.
[60] H. Mao, S. Chen, D. Dimmery, S. Singh, D. Blaisdell, Y. Tian, M. Alizadeh, and E. Bakshy, "Real-world video adaptation with reinforcement learning," in *ICML Reinforcement Learning for Real Life Workshop*, 2019.
[61] S. M. Kakade and A. Tewari, "On the generalization ability of online strongly convex programming algorithms," in *Proc. NIPS*, 2009, pp. 801–808.

APPENDIX: PROOFS OF THEOREMS

We begin by proving that the loss function of PiTree is LIpschitz [61] and STrongly convex (LIST):

Lemma 1. *$\ell(r; r^*)$ in Equation 2 is Lipschitz, i.e. $\exists \mathcal{L}$, s.t. $\forall r^*, r_1, r_2 \in [R_{min}, R_{max}]$,*

$$|\ell(r_1; r^*) - \ell(r_2; r^*)| \leqslant \mathcal{L} \cdot |r_1 - r_2| \tag{7}$$

Lemma 2. *$\ell(r; r^*)$ in Equation 2 is strongly convex, i.e. $\exists \nu$, s.t. $\forall r^*, r_1, r_2 \in [R_{min}, R_{max}], \forall \lambda \in [0, 1]$,*

$$\ell\left(\lambda r_1 + (1-\lambda) r_2; r^*\right) \leqslant \lambda \ell(r_1; r^*) + (1-\lambda)\ell(r_2; r^*) - \left(\frac{\nu}{2}\right)\lambda(1-\lambda)(r_1 - r_2)^2 \tag{8}$$

Proof of Lemma 1. $\forall r^*, r_1, r_2 \in [R_{min}, R_{max}]$, we have

$$\begin{aligned} |\ell(r_1; r^*) - \ell(r_2; r^*)| &= \frac{|(r_1 - r^*)^2 - (r_2 - r^*)^2|}{(R_{max} - R_{min})^2} = \frac{|r_1 + r_2 - 2r^*| \cdot |r_1 - r_2|}{(R_{max} - R_{min})^2} \leqslant \frac{(|r_1 - r^*| + |r_2 - r^*|) \cdot |r_1 - r_2|}{(R_{max} - R_{min})^2} \\ (\because r^*, r_1, r_2 \in [R_{min}, R_{max}]) \quad &\leqslant \left(\frac{2}{R_{max} - R_{min}}\right)|r_1 - r_2| \end{aligned} \tag{9}$$

Thus $\ell(r; r^*)$ is Lipschitz with Lipschitz constant $\mathcal{L} = 2/(R_{max} - R_{min})$. □

Proof of Lemma 2. Similarly, we could also demonstrate that $\ell(r; r^*)$ is strongly convex: $\forall \lambda \in [0, 1]$, we have:

$$\begin{aligned} \ell(\lambda r_1 + (1-\lambda) r_2; r^*) &= \frac{\left(\lambda(r_1 - r^*) + (1-\lambda)(r_2 - r^*)\right)^2}{(R_{max} - R_{min})^2} \\ &= \lambda^2 \ell(r_1; r^*) + (1-\lambda)^2 \ell(r_2; r^*) + \frac{2\lambda(1-\lambda)(r_1 - r^*)(r_2 - r^*)}{(R_{max} - R_{min})^2} \\ &= \lambda \ell(r_1; r^*) + (1-\lambda)\ell(r_2; r^*) - \lambda(1-\lambda)\left(\ell(r_1; r^*) + \ell(r_2; r^*) - \frac{2(r_1 - r^*)(r_2 - r^*)}{(R_{max} - R_{min})^2}\right) \\ &= \lambda \ell(r_1; r^*) + (1-\lambda)\ell(r_2; r^*) - \frac{1}{(R_{max} - R_{min})^2}\lambda(1-\lambda)\left(r_1 - r_2\right)^2 \end{aligned} \tag{10}$$

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

With the LIST loss function, the two theorems in §IV could be derived from two existing papers on the performance bound of teacher-student learning [33, 61].

Proof of Theorem 1 (§IV-A). From the COROLLARY 5 in [61], we have

$$Reg_T := \sum_{t=1}^{T} \ell\left(r^{(t)}; r^{*(t)}\right) - \min_{r \in [R_{min}, R_{max}]} \sum_{t=1}^{T} \ell\left(r; r^{*(t)}\right) \leqslant \frac{\mathcal{L}^2(1 + \log T)}{2\nu} \tag{11}$$

where $\mathcal{L}$ and ν are the constants defined above, and Reg_T is the *regret* of the algorithm if it did not take the optimal decision. As for PiTree,

$$\because r^{*(t)} \in [R_{min}, R_{max}], \quad \min_{r \in [R_{min}, R_{max}]} \sum_{t=1}^{T} \ell\left(r; r^{*(t)}\right) = \min_{r \in [R_{min}, R_{max}]} \sum_{t=1}^{T} \left(r - r^{*(t)}\right)^2 = 0 \tag{12}$$

Substituting $\mathcal{L}$ and ν in Equation 11 with the expressions from Equation 9 and 10, we have:

$$\frac{1}{T}\sum_{t=1}^{T}\left(\hat{r}^{(t)} - r^{*(t)}\right)^2 \leqslant \frac{1 + \log T}{T}\left(R_{max} - R_{min}\right)^2 \tag{13}$$

Thus, the RMSE is bounded after rooting the equation above. □

Proof of Theorem 2 (§IV-B). Since the loss function $\ell(r; r^*)$ satisfies the LIST assumption, and also the output actions of ABR algorithms (bitrates) are discrete, we could extend the THEOREM 3.3 introduced in [33] with THEOREM 2 in [61]. Let $Q_t^{\pi'}(s, \pi)$ denote the t-step cost of executing action a in initial state s and then following policy π':

$$Q_t^{\pi'}(s, a) = \frac{1}{(R_{max} - R_{min})^2}\left(\left(a - \pi^*(s_1)\right)^2 + \sum_{\tau=2}^{t}\left(\pi'(s_\tau) - \pi^*(s_\tau)\right)^2\right) \tag{14}$$

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

$$Q^{\pi*}_{T-t+1}(s,a) - Q^{\pi*}_{T-t+1}(s,\pi^*(s)) = \frac{(a-\pi^*(s))^2}{(R_{max}-R_{min})^2} \leqslant 1 \triangleq u \tag{15}$$

Hence the proof follows [33] and [61] with the fact that $u = 1$. □

Zili Meng (S'18) received the B.Eng. degree in Department of Electronic Engineering at Tsinghua University in 2019. He is currently pursuing the Ph.D. degree with Institute for Network Science and Cyberspace, Tsinghua University. He has published papers in ACM SIGCOMM, ACM Multimedia, IEEE JSAC and so on. His research interests include learning-based networked systems and video streaming. He was a recipient of Microsoft Research Asia Fellowship in 2020, also the winner of the Student Research Competition in ACM SIGCOMM 2018.

Yaning Guo is currently pursuing the bachelor's degree with the Department of Electronic Engineering, Tsinghua University. She has coauthored paper in ACM Multimedia. Her research interests include networked system and related algorithms.

Yixin Shen is currently pursuing the bachelor's degree with the Department of Electronic Engineering, Tsinghua University. His research interests include video streaming and related algorithms.

Jing Chen received the B.Eng. degree in Department of Electronic Engineering at Tsinghua University in 2020. She is currently pursuing a master's degree in the Institute for Network Sciences and Cyberspace, Tsinghua University. She is highly interested in learning-based networked systems and multimedia streaming technology, and has a coauthored paper in ACM Multimedia.

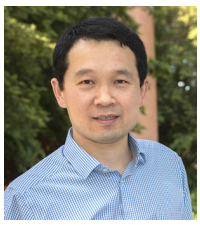
Chao Zhou received his Ph.D. degree from the Institute of Computer Science and Technology, Peking University, Beijing, China, in 2014. He has been with Beijing Kuaishou Technology Co., Ltd. as an Algorithm Scientist. Dr. Zhou's research interests include HTTP video streaming, joint source-channel coding,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

Minhu Wang received the B.Eng. degree in Department of Electronic Engineering at Tsinghua University in 2019. He is currently pursuing the Ph.D. degree with Institute for Network Science and Cyberspace, Tsinghua University. His research interests include learning-based network systems and network function virtualization.

Jia Zhang received the B.Eng. degree in Department of Electronic Engineering at Tsinghua University in 2018. She is currently pursuing the Ph.D. degree with Department of Computer Science and Technology, Tsinghua University. Her research interests include learning-based network systems and transport layer.

Mingwei Xu (M'03) is a full professor of the Department of Computer Science and Technology in Tsinghua University. He is a member of technical steering committee of China Communications Standard Association (CCSA). His research interests include future Internet architecture, Internet routing and network virtualization. He has chaired or participated in more than 30 research projects, and published over 200 papers. He has won the second prize of national scientific and technological progress award for three times and the second prize of national technology invention award once. He is the winner of National Science Foundation for Distinguished Young Scholars of China. He has served as TPC chair or member for several IEEE conferences, such as ICPP, Globecom and ICC.

Chen Sun received the B.S. degree from the Department of Electronic Engineering, Tsinghua University, Beijing, China, in 2014, and the Ph.D. degree from the Department of Computer Science and Technology, Tsinghua University, Beijing, China, in 2019. He is currently an Engineer in Alibaba Group. He has published papers in SIGCOMM, JSAC, ToN, INFOCOM, ICNP, and so on. His research interests include software-defined network, network function virtualization, and network monitoring. He was a recipient of Google PhD Fellowship Award in 2018.



Hongxin Hu (S'10–M'12) is an Associate Professor in the Department of Computer Science and Engineering, University at Buffalo, the State University of New York. He received the Ph.D. degree in computer science from Arizona State University, Tempe, AZ, in 2012. His current research interests include security in emerging networking technologies, security in Internet of Things (IoT), security and privacy in social networks, and security in cloud and mobile computing. He has published over 100 refereed technical papers, many of which appeared in top conferences and journals. He received the NSF CAREER Award in 2019. He is the recipient of the Best Paper Awards from ACSAC 2020, ACM SIGCSE 2018, and ACM CODASPY 2014, and the Best Paper Award Honorable Mentions from ACM SACMAT 2016, IEEE ICNP 2015, and ACM SACMAT 2011.