Neural Adaptive Video Streaming with Pensieve

by

Hongzi Mao

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2017

Signature redacted

Author .....
Department of Electrical Engineering and Computer Science
May 19, 2017

Signature redacted

Certified by..
Mohammad Alizadeh
Assistant Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Signature redacted

Accepted by .....
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Neural Adaptive Video Streaming with Pensieve

by

Hongzi Mao

Submitted to the Department of Electrical Engineering and Computer Science
on May 19, 2017, in partial fulfillment of the
requirements for the degree of
Master of Science in Electrical Engineering and Computer Science

Abstract

Client-side video players employ bitrate adaptation algorithms to cater to the ever-growing QoE requirements of users. These ABR algorithms must balance multiple QoE factors, such as maximizing video bitrate and minimizing rebuffering times. Despite the abundance of recently proposed ABR algorithms, state-of-the-art schemes suffer from two practical challenges: (1) throughput prediction is difficult and inaccurate predictions can lead to degraded performance; (2) existing algorithms use fixed heuristics which have been fine-tuned according to strict assumptions about deployment environments — such tuning precludes generalization across network conditions and QoE objectives.

To overcome these challenges, we develop Pensieve, a system that generates ABR algorithms entirely using Reinforcement Learning (RL). Pensieve uses RL to train a neural network model that selects bitrates for future video chunks based on observations collected by client video players. Unlike existing approaches, Pensieve does not rely upon pre-programmed models or assumptions about the environment. Instead, it learns to make ABR decisions solely through observations of the resulting performance of past decisions. As a result, Pensieve can automatically learn ABR algorithms that adapt to a wide range of environmental conditions and QoE metrics. We compare Pensieve to state-of-the-art ABR algorithms using trace-driven and real world experiments spanning a wide variety of network conditions, QoE metrics, and video properties. In all considered scenarios, Pensieve outperforms the best state-of-the-art scheme, with improvements in average QoE of 13.1%–25.0%. Pensieve's policies generalize well, outperforming existing schemes even on networks on which it was not trained.

Thesis Supervisor: Mohammad Alizadeh
Title: Assistant Professor of Electrical Engineering and Computer Science

Acknowledgments

This research was performed under the supervision of Professor Mohammad Alizadeh, and it was published in SIGCOMM 2017. My first year of graduate studies was generously supported by the Andrew (1956) and Erna Viterbi Fellowship. In addition, this work was partially supported by the National Science Foundation.

I would like to thank Mohammad for epitomizing what an ideal advisor, mentor, researcher and friend would be. I am truly blessed to have him open my eyes to the landscape of research problems, guide me towards a uniquely interesting path and support me perpetually through all difficulties. It fascinates me every time when I think about all the adventures we are going to take in the upcoming years.

I also want to thank Professor Dina Katabi and Professor Fadel Adib for introducing me to the wizard world of MIT. They taught me the hard working spirit and the desire of advancing the state-of-the-art. I am tremendously indebted to Dina for encouraging me to explore and pursue the broader areas in Computer Systems and Machine Learning.

I am lucky to be surrounded by the most brilliant and nicest students in NMS and NETMIT groups. In particular, I thank Ravi to take me on such a wonderful journey of designing, implementing and deploying Pensieve. I am also grateful to have Peter, Prateesh and Mehrdad as my officemates who constantly give me so much joy and fun every day. I also thank all my friends both here in Cambridge and back home.

It has also been an unforgettable experience with Srikanth and Ishai during my internship at Microsoft Research, especially for the distributed writing collaboration of DeepRM, where all of us were literally in different places of the world.

I am thankful to have Jiaming, Changchen and Feihu as my roommates who make it a home when we are away from home. Also fortunately, many late-night discussions with Jiaming have become parts of the foundations of Pensieve and DeepRM.

Lastly but most importantly, I thank my parents Jianbo and Liyu, and my girlfriend Yujie. Their unconditional support and love have made the other end of the world feel like just seconds away. Finally, I turn to my mother. Words can not express my gratitude. Thank you for making me who I am and thank you for always being there for me; this thesis is dedicated to you.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Recent years have seen a rapid increase in the volume of HTTP-based video streaming traffic [2, 3]. Concurrent with this increase has been a steady rise in user demands on video quality. Many studies have shown that users will quickly abandon video sessions if the quality is not sufficient, leading to significant losses in revenue for content providers [4, 5]. Nevertheless, content providers continue to struggle with delivering high-quality video to their viewers.

Adaptive bitrate algorithms (ABR) are the primary tool that content providers use to optimize video quality. These algorithms run on client-side video players and dynamically choose a bitrate for each video *chunk* (e.g., 4-second block). ABR algorithms make bitrate decisions based on various observations such as the estimated network throughput and playback buffer occupancy. Their goal is to maximize the user's quality of experience (QoE) by adapting the video bitrate to the current network conditions. However, selecting the right bitrate is challenging in practice for several reasons:

1. **Network variability:** ABR algorithms must deal with inaccurate throughput predictions [6, 7, 8] because networks can be highly variable and unpredictable, e.g., time-varying cellular networks [9, 10].
2. **Conflicting and diverse QoE requirements:** QoE objectives (e.g., high bitrates, minimal rebuffering, smoothness) are inherently at odds with one another. Moreover, users have different QoE preferences [11, 12, 13]; some want the highest bitrate, while others are intolerant to rebuffering, or prefer smooth

playback.

3. **Coarse-grained control decisions:** videos are typically encoded at a handful of bitrates [14]; ABR algorithms are thus restricted to a small set of choices and cannot make fine-grained control decisions.
4. **Long-term effects of decisions:** bitrate decisions for one video chunk can have cascading effects on later chunks (e.g., when smoothness is considered). As a result, ABR algorithms should proactively plan for future chunks rather than only the next immediate chunk.

We elaborate on these challenges in §2.

Many ABR algorithms have been proposed in recent years (§3). The majority of existing algorithms develop fixed control rules for making bitrate decisions based on estimated network throughput ("rate-based" algorithms [15, 7]), playback buffer size ("buffer-based" schemes [16, 17]), or a combination of the two signals [18]. These schemes require significant hand-tuning and do not generalize to different network conditions and QoE objectives.

Recently, MPC [1] proposed a more sophisticated approach which makes bitrate decisions by solving a QoE maximization problem over a horizon of several future chunks. By optimizing directly for the desired QoE objective, MPC can perform better than approaches which use fixed heuristics. However, MPC requires an accurate model of the system dynamics. As we show, this makes MPC sensitive to inaccurate throughput predictions and the length of the optimization horizon (§4).

In this thesis, we propose *Pensieve*,[1] a system that learns ABR algorithms automatically, without using any pre-programmed models or explicit assumptions about the operating environment. Pensieve uses modern Reinforcement Learning (RL) techniques [20, 21, 22] to learn a control policy for bitrate adaptation *purely through experience*. During training, Pensieve starts knowing nothing about the task at hand. It then gradually learns to make better ABR decisions through reinforcement, in the form of reward signals that reflect video QoE for past decisions. By optimizing its control policy based on the actual performance of past choices, Pensieve discovers policies that are much more robust than those used by algorithms which rely on accurate system models.

[1] A pensieve is a device used in Harry Potter [19] to review memories.

Pensieve represents its control policy as a neural network that maps "raw" observations (e.g., past throughput samples, playback buffer occupancy, video chunk sizes) to the bitrate decision for the next chunk. Pensieve trains this neural network using A3C [20], a state-of-the-art online actor-critic RL algorithm. We describe the basic training algorithm and present extensions that allow it to generalize to support videos with different properties, e.g., bitrate levels (§5).

To train its models, Pensieve uses simulations over a large corpus of network traces. While Pensieve can also train with data collected from live video clients (§7), simulations are much faster and allow Pensieve to "experience" 100 hours of video downloads in only 10 minutes. We show in §5.1 that Pensieve's simulation environment faithfully models video streaming with live video players.

We evaluate Pensieve using a full system implementation (§5.4). Our implementation deploys Pensieve's neural network model on an *ABR server*, which video clients query to get the bitrate to use for the next chunk; client requests include observations about throughput, buffer occupancy, and video properties. This design removes the burden of performing neural network computation on video clients, which may have limited computation power, e.g., TVs, mobile devices, etc. (§7).

We compare Pensieve to state-of-the-art ABR algorithms using a broad set of network conditions (both with trace-based emulation and in the wild) and QoE metrics (§6.2). We find that in all considered scenarios, Pensieve is able to rival or outperform the best existing scheme, with average QoE improvements ranging from 13.1%–25.0%. Additionally, our results illustrate Pensieve's ability to generalize across both unseen network conditions and video properties. For example, on both broadband and HSDPA networks, Pensieve was able to outperform all existing ABR algorithms by training solely with a synthetic dataset. Finally, we present results which highlight Pensieve's low overhead and lack of sensitivity to certain system parameters, e.g., in the neural network (§6.4).

Chapter 2

Background

HTTP-based adaptive streaming (standardized as DASH [23]) is the predominant form of video delivery today. By transmitting video using HTTP, content providers are able to leverage existing CDN infrastructure and maintain simplified (stateless) backends. Further, HTTP is compatible with a multitude of client-side applications such as web browsers and mobile applications.

In DASH systems, videos are stored on servers as multiple chunks, each of which represents a few seconds of the overall video playback. Each chunk is encoded at several discrete bitrates, where a higher bitrate implies a higher quality and thus a larger chunk size. Chunks across bitrates are aligned to support seamless quality transitions, i.e., a video player can switch to a different bitrate at any chunk boundary without fetching redundant bits or skipping parts of the video.

Figure 2-1 illustrates the end-to-end process of streaming a video over HTTP today. As shown, a player embedded in a client application first sends a token to a video service provider for authentication. The provider responds with a manifest file that directs the client to a CDN hosting the video and lists the available bitrates for the video. The client then requests video chunks one by one, using an *Adaptive Bitrate (ABR) algorithm.* These algorithms use a variety of different inputs (e.g., playback buffer occupancy, throughput measurements, etc.) to select the bitrate for future chunks. As chunks are downloaded, they are played back to the client; note that playback of a given chunk cannot begin until the entire chunk has been downloaded.

Challenges: The policies employed by ABR algorithms heavily influence video

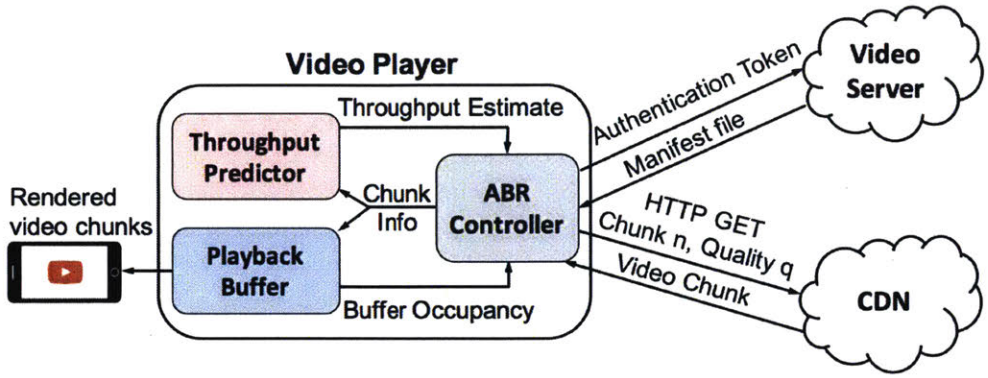


Figure 2-1: An overview of HTTP adaptive video streaming.

streaming performance. However, these algorithms face four primary practical challenges:

1. ABR algorithms must balance a variety of QoE goals such as maximizing video quality (i.e., highest average bitrate), minimizing rebuffering events (i.e., scenarios where the client's playback buffer is empty), and maintaining video quality smoothness (i.e., avoiding constant bitrate fluctuations). However, many of these goals are inherently conflicting [24, 6, 15]. For example, on networks with limited bandwidth, consistently requesting chunks encoded at the highest possible bitrate will maximize quality, but may increase rebuffer rates. Conversely, on varying networks, choosing the highest bitrate that the network can support at any time could lead to substantial quality fluctuation, and hence degraded smoothness. To further complicate matters, preferences among these QoE factors vary significantly across users [11, 12, 25, 13].
2. Network conditions can fluctuate over time and can vary significantly across environments. This complicates bitrate selection as different scenarios may require different weights for input signals. For example, on time-varying cellular links, throughput prediction is often inaccurate and cannot account for sudden fluctuations in network bandwidth—inaccurate predictions can lead to underutilized networks (lower video quality) or inflated download delays (rebuffering). To overcome this, ABR algorithms must prioritize more stable input signals like buffer occupancy in these scenarios.
3. Bitrate selection for a given chunk can have cascading effects on the state of the video player. For example, selecting a high bitrate may deplete the play-

back buffer and force subsequent chunks to be downloaded at low bitrates (to avoid rebuffering). Additionally, a given bitrate selection will directly influence the next decision when smoothness is considered—ABR algorithms will be less inclined to change bitrates.

4. The control decisions available to ABR algorithms are coarse-grained as they are limited to the available bitrates for a given video. Thus, there exist scenarios where the estimated throughput falls just below one bitrate, but well above the next available bitrate. In these cases, the ABR algorithm must decide whether to prioritize higher quality or the risk of rebuffering.

Chapter 3

Related Work

Many ABR algorithms have been proposed to address these challenges. The earliest solutions can be primarily grouped into two classes: rate-based and buffer-based. Rate-based algorithms [15, 7] first estimate the available network bandwidth using past chunk downloads, and then request chunks at the highest bitrate that the network is predicted to support. However, these methods are hindered by the biases present when estimating available bandwidth on top of HTTP [26, 18]. Several systems aim to correct these throughput estimates using smoothing heuristics and data aggregation techniques [7], but accurate throughput prediction remains a challenge in practice [8]. In contrast, buffer-based approaches [16, 17] solely consider the client's playback buffer occupancy when deciding the bitrates for future chunks. The goal of these algorithms is to keep the buffer occupancy at a pre-configured level which balances rebuffering and video quality. The most recent buffer-based approach, BOLA [17], optimizes for a specified QoE metric using a Lyapunov optimization based entirely on buffer occupancy observations.

Each of these approaches performs well in certain settings but not in others. Specifically, rate-based approaches are best at startup time and when link rates are stable, while buffer-based approaches are sufficient and more robust in steady state and in the presence of time-varying networks [16]. Consequently, recently proposed ABR algorithms have also investigated combining these two techniques. The state-of-the-art approach is MPC [1], which employs model predictive control algorithms that use both throughput estimates and buffer occupancy information to select bi-

trates that are expected to maximize QoE over a horizon of several future chunks. However, MPC still relies heavily on accurate throughput estimates which are not always available. When throughput predictions are incorrect, MPC's performance can degrade significantly. Addressing this issue requires heuristics that make throughput predictions more conservative. However, tuning such heuristics to perform well in different environments is challenging. Further, as we will describe in §4, MPC is often unable to plan far enough into the future to apply the policies that would maximize performance in given settings.

The closest previous work of using Reinforcement Learning in adaptive video streaming are [27] and [28]. These work propose a similar formulation where an RL agent gradually improves its bitrate adaptation policy by interacting with the video streaming environment. However, these systems are largely simulation based, which only preliminarily demonstrate the "learnability" of an RL policy. Also, recent developments in RL algorithms [20, 29, 30, 31] have significantly improved the classic algorithms used in these work. In contrast, we integrate the modern learning system into an industry standard DASH player [23] (§5) and comprehensively compare the performance with state-of-the-art approaches (§6).

Summary: Existing classic ABR algorithms all rely on fixed heuristics that have been exhaustively tuned according to rigid assumptions about deployment environments. Further, each of these schemes has been explicitly configured to optimize for a specific QoE metric. As a result, today's ABR algorithms fail to generalize, and experience degraded performance if any assumptions are violated. In these scenarios, schemes must again be manually tuned to cater to the new environment. To overcome these issues, we turn to a learning-based approach for generating ABR algorithms. In the next section, we will describe our high-level approach and explain why this technique can overcome the aforementioned challenges to produce robust ABR algorithms.

Chapter 4

Learning ABR Algorithms

In this thesis, we consider a learning-based approach to generating ABR algorithms. Unlike prior approaches which use preset rules in the form of fine-tuned heuristics (§3), our technique attempts to learn an ABR policy from observations. Specifically, our approach is based on Reinforcement Learning (RL). RL considers a general setting in which an *agent* interacts with an *environment*. At each time step t, the agent observes some *state* s_t, and chooses an *action* a_t. After applying the action, the state of the environment transitions to s_{t+1} and the agent receives a *reward* r_t. The goal of learning is to maximize the expected cumulative discounted reward: $\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$, where $\gamma \in (0, 1]$ is a factor discounting future rewards.

Figure 4-1 summarizes how RL can be applied to bitrate adaptation. As shown, the decision policy guiding the ABR algorithm is not handcrafted. Instead, it is derived from training a neural network. The ABR agent observes a set of metrics including the client playback buffer occupancy, past bitrate decisions, and several raw

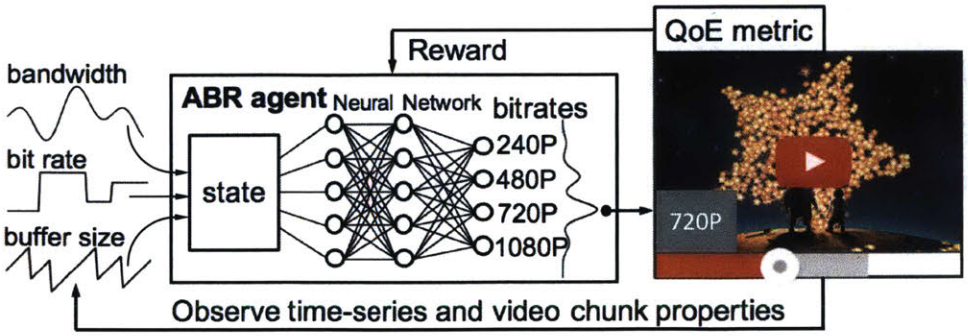
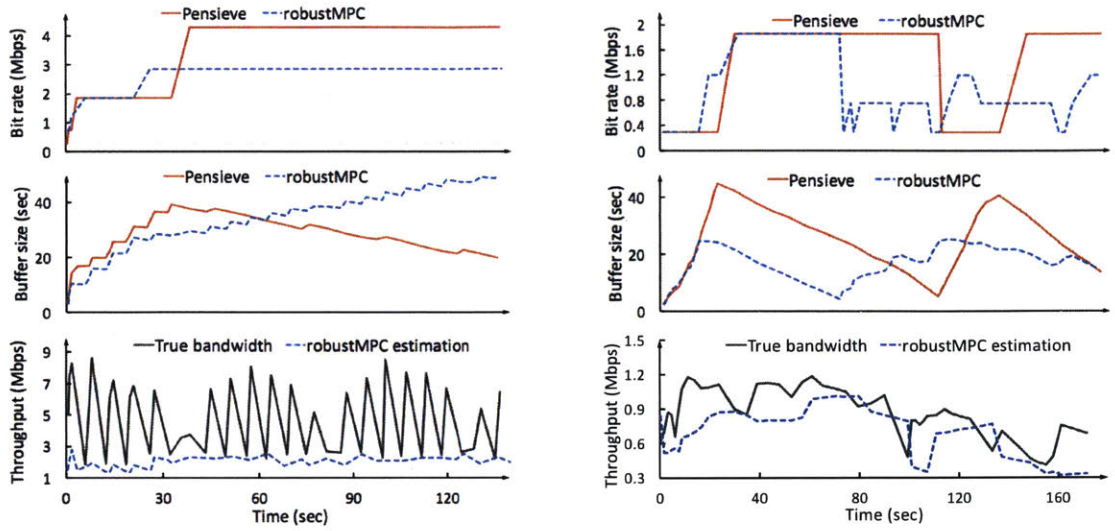


Figure 4-1: Applying Reinforcement Learning to bitrate adaptation.



(a) Synthetic network.

(b) HSDPA network.

Figure 4-2: Profiling bitrate selections, buffer occupancy, and throughput estimates with robustMPC [1] and Pensieve.

network signals (e.g., throughput measurements) and feeds these values to the neural network, which outputs the action, i.e., the bitrate to use for the next chunk. The resulting QoE is then observed and passed back to the ABR agent as a reward. The agent uses the reward information to train and improve its neural network model. More details about the specific training algorithms we used are provided in §5.2.

To motivate learning-based ABR algorithms, we now provide two examples of situations where existing techniques that rely on fixed heuristics can perform poorly. We choose these examples for illustrative purposes. We do not claim that they are indicative of the performance gains with learning in realistic network scenarios. We perform thorough quantitative evaluations comparing learning-generated ABR algorithms to existing schemes in §6.2.

In these examples, we compare RL-generated ABR algorithms to MPC [1]. Recall from §3 that MPC uses both throughput estimates and observations about buffer occupancy to select bitrates that maximize a given QoE metric across a future chunk horizon. Here we consider robustMPC, a version of MPC that is configured to use a horizon of 5 chunks, and a conservative throughput estimate which normalizes the default throughput prediction with the max prediction error over the past 5 chunks. As the MPC paper shows, and our results validate, robustMPC's conservative through-

put prediction significantly improves performance over default MPC, and achieves a high level of performance in most cases (§6.2). However, heuristics like robustMPC's throughput prediction require careful tuning and can backfire when their design assumptions are violated.

Example 1: The first example considers a scenario in which the network throughput is highly variable. Figure 4-2a compares the network throughput specified by the input trace with the throughput estimates used by robustMPC. As shown, robustMPC's estimates are overly cautious, hovering around 2 Mbps instead of the average network throughput of roughly 4.5 Mbps. These inaccurate throughput predictions prevent robustMPC from reaching high bitrates even though the occupancy of the playback buffer continually increases. In contrast, the RL-generated algorithm is able to properly assess the high average throughput (despite fluctuations) and switch to the highest available bitrate once it has enough cushion in the playback buffer. The RL-generated algorithm considered here was trained on a large corpus of real network traces (§6.1), not the synthetic trace in this experiment. Yet, it was able to make the appropriate decision.

Example 2: In our second example, both robustMPC and the RL-generated ABR algorithm optimize for a new QoE metric which is geared towards users who strongly prefer HD video. This metric assigns high reward to HD bitrates and low reward to all other bitrates, while still favoring smoothness and penalizing rebuffering. To optimize for this metric, an ABR algorithm should attempt to build the client's playback buffer to a high enough level such that the player can switch up to and maintain an HD bitrate level. Using this approach, the video player can maximize the amount of time spent streaming HD video, while minimizing rebuffering time and bitrate transitions. However, performing well in this scenario requires long term planning since at any given instant, the penalty of selecting a higher bitrate (HD or not) may be incurred many chunks in the future when the buffer cannot support multiple HD downloads.

Figure 4-2b illustrates the bitrate selections made by each of these algorithms, and the effects that these decisions have on the playback buffer. Note that robustMPC and the RL-generated algorithm were both configured to optimize for this new QoE metric. As shown, robustMPC is unable to apply the aforementioned policy. Instead, robustMPC maintains a medium-sized playback buffer and requests chunks at bitrates

that fall between the lowest level (300 kbps) and the lowest HD level (1850 kbps). The reason is that, despite being tuned to consider a horizon of future chunks at every step, robustMPC fails to plan far enough into the future. In contrast, the RL-generated ABR algorithm is able to actively implement the policy outlined above. It quickly grows the playback buffer by requesting chunks at 300 kbps, and then immediately jumps to the HD quality of 1850 kbps; it is able to then maintain this level for nearly 80 seconds, thereby ensuring quality smoothness.

Chapter 5

Design

In this section, we describe the design and implementation of Pensieve, a concrete system that generates RL-based ABR algorithms and applies them to video streaming sessions. We start by explaining the training methodology (§5.1) and algorithms (§5.2) underlying Pensieve. We then describe an important enhancement that Pensieve makes to existing RL algorithms, which enables it to support different videos using a single model (§5.3). Finally, we explain the implementation details of Pensieve and how it applies learned models to real streaming sessions (§5.4).

5.1 Training Methodology

The first step of Pensieve is to generate an ABR algorithm using RL (§4). To do this, Pensieve runs a training phase in which the learning agent explores a video streaming environment. Ideally, training would occur using actual video streaming clients. However, emulating the standard video streaming environment entails using a web browser to continually download video chunks. This approach is slow, as the training algorithm must wait until all of the chunks in a video are completely downloaded before updating its model.

To accelerate this process, Pensieve trains ABR algorithms in a simulation environment that faithfully models the dynamics of video streaming with real client applications. Pensieve's simulator maintains an internal representation of the client's playback buffer. For each chunk download, the simulator assigns a download time

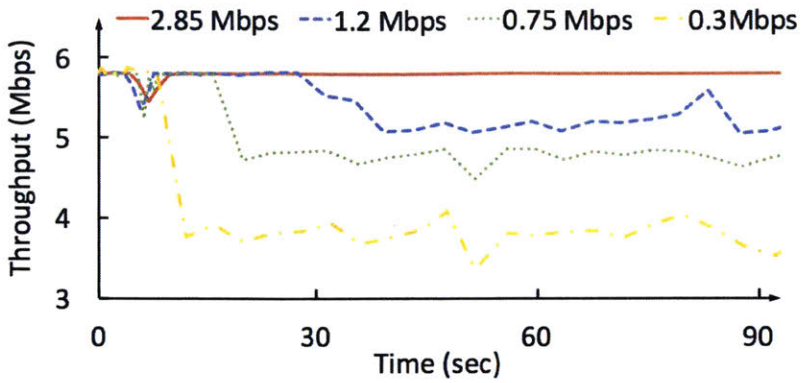
that is solely based on the chunk's bitrate and the input network throughput traces. The simulator then drains the playback buffer by the current chunk's download time, to represent video playback during the download, and adds the playback duration of the downloaded chunk to the buffer. The simulator carefully keeps track of rebuffering events that arise as the buffer occupancy changes, i.e., scenarios where the chunk download time exceeds the buffer occupancy at the start of the download. In scenarios where the playback buffer cannot accommodate video from an additional chunk download, Pensieve's simulator pauses requests for 500 ms before retrying.[1] After each chunk download, the simulator passes several state observations to the RL agent for processing: the current buffer occupancy, rebuffering time, chunk download time, size of the next chunk (at all bitrates), and the number of remaining chunks in the video. We describe how this input is used by the RL agent in more detail in §5.2. Using this simulation approach, Pensieve can "experience" 100 hours of video downloads in only 10 minutes.

Though modeling the application layer semantics of client video players is straightforward, faithful simulation is complicated by intricacies at the transport layer. Specifically, video players may not request future chunks as soon as a chunk download completes, e.g., because the playback buffer is full. Such delays can trigger the underlying TCP connection to revert to slow start, a behavior known as *slow-start-restart* [32]. Slow start may in turn prevent the video player from fully using the available bandwidth, particularly for small chunk sizes (low bitrates). This behavior makes simulation challenging as it inherently ties network throughput to the ABR algorithm being used, e.g., schemes that fill buffers quickly will experience more slow start phases and thus less network utilization.

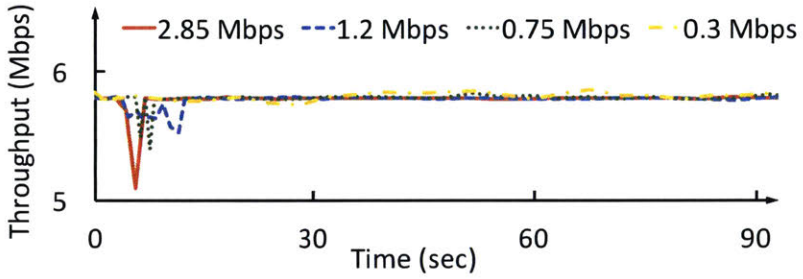
To verify this behavior, we loaded the test video described in §6.1 over an emulated 6 Mbps link using four ABR algorithms, each of which continually requests chunks at a single bitrate. We loaded the video with each scheme twice, both with slow-start-restart enabled and disabled.[2] Figure 5-1 shows the throughput usage *during chunk downloads* for each bitrate in both scenarios. As shown, with slow-start-restart enabled, the throughput depends on the bitrate of the chunk; ABR algorithms using

[1] This is the default request retry rate used by DASH players [23].

[2] In Linux, this can be configured using the `net.ipv4.tcp_slow_start_after_idle` parameter.



(a) With TCP slow start restart enabled.



(b) With TCP slow start restart disabled.

Figure 5-1: Profiling the throughput usage per-chunk of commodity video players with and without TCP slow start restart.

lower bitrates (smaller chunk sizes) achieve less throughput per chunk. However, throughput is consistent and matches the available bandwidth (6 Mbps) for different bitrates if we disable slow-start-restart.

Pensieve's simulator assumes that the throughput specified by the trace is entirely used by each chunk download. As the above results show, this can be achieved by disabling slow-start-restart on the video server. Disabling slow-start-restart could increase traffic burstiness, but recent standards efforts are tackling the same problem for video streaming more gracefully by pacing the initial burst from TCP following an idle period [33, 34].

In the end, no simulation can capture all real world system artifacts with 100% accuracy. However, we find that Pensieve can learn very high quality ABR algorithms (§6.2) using imperfect simulations, as long as it experiences a large enough variety of network conditions during training.

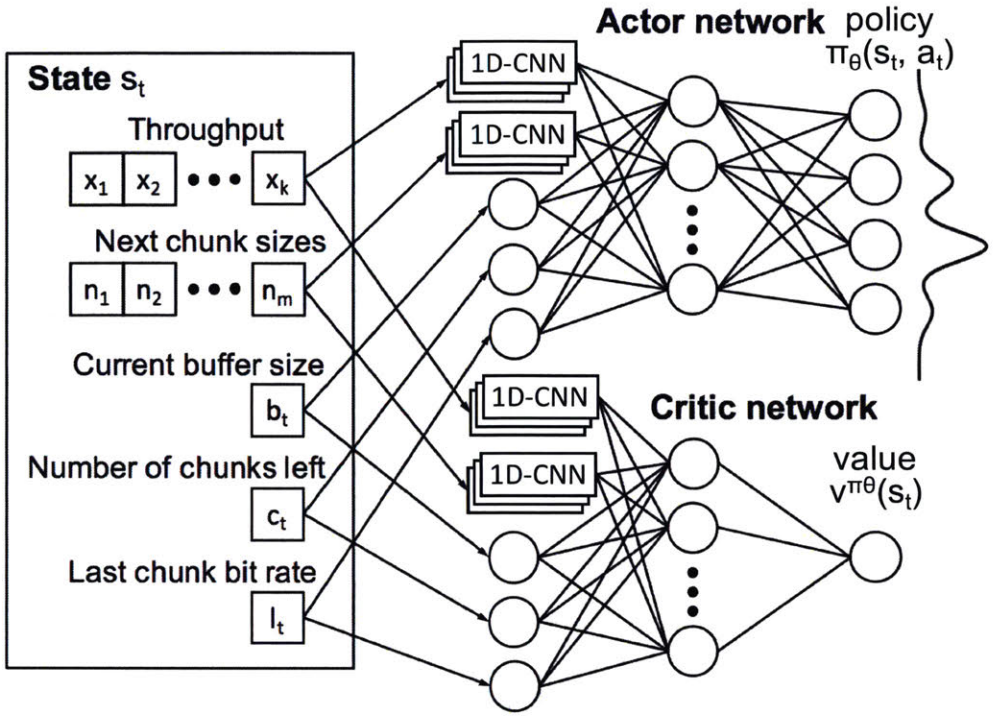


Figure 5-2: The Actor-Critic algorithm that Pensieve uses to generate ABR policies (described in §5.4).

5.2 Basic Training Algorithm

We now describe our training algorithms. As shown in Figure 5-2, Pensieve's training algorithm uses A3C [20], a state-of-the-art actor-critic method which involves training two neural networks. The detailed functionalities of these networks are explained in below.

Inputs: After download of each chunk t, Pensieve's learning agent takes state inputs $s_t = (\vec{x_t}, \vec{n_t}, b_t, c_t, l_t)$ to the neural networks, where $\vec{x_t}$ is the network throughput measurements for the past k video chunks; $\vec{n_t}$ is a vector of m available sizes for the next video chunk; b_t is the current buffer level; c_t is the number of chunks remaining in the video; and l_t is the bit rate at which the last chunk was downloaded.

Policy: Upon receiving s_t, Pensieve's RL agent needs to take an action a_t that corresponds to the bit rate for the next video chunk. The agent selects actions based on a *policy*, defined as a probability distribution over actions $\pi : \pi(s_t, a_t) \rightarrow [0, 1]$. $\pi(s_t, a_t)$ is the probability that action a_t is taken in state s_t. In practice, there are intractably many {state, action} pairs, e.g., throughput estimates and buffer

occupancies are continuous real numbers. To overcome this, Pensieve uses a Neural Network (NN) [35] to represent the policy with a manageable number of adjustable parameters, θ, which we refer to as policy parameters. Using θ, we can represent the policy as $\pi_\theta(s_t, a_t)$. NNs have recently been applied successfully to solve large-scale RL tasks [29, 30, 22]. An advantage of NNs is that they do not need hand-crafted features, and can be applied directly to "raw' observation signals. The *actor network* in Figure 5-2 depicts how Pensieve uses an NN to represent an ABR policy. We describe how we design the specific architecture of the NN in §6.3.

Policy gradient methods: After applying each action, the simulated environment provides the learning agent with a reward r_t for that chunk. Recall from §4 that the primary goal of the RL agent is to maximize the expected cumulative (discounted) reward that it receives from the environment. Thus, the reward is set to reflect the performance of each chunk download according to the specific QoE metric we wish to optimize. See §6 for examples of QoE metrics.

The actor-critic algorithm used by Pensieve to train its policy is an instance of *policy gradient methods* [36]. The key idea in policy gradient methods is to estimate the gradient of the expected total reward by observing the trajectories of executions obtained by following the policy. The gradient of the cumulative discounted reward with respect to the policy parameters, θ, can be computed as [20]:

$$\nabla_\theta \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right] = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) A^{\pi_\theta}(s,a)\right]. \tag{5.1}$$

$A^{\pi_\theta}(s,a)$ is the *advantage* function, which represents the difference in the expected total reward when we *deterministically* pick action a in state s, compared with the expected reward for actions drawn from policy π_θ. The advantage function encodes how much better a specific action is compared to the "average" action taken according to the policy.

In practice, the agent samples a trajectory of bitrate decisions and uses the empirically computed the advantage $A(s_t, a_t)$, as an unbiased estimate of $A^{\pi_\theta}(s_t, a_t)$. Each

update of the actor network parameter θ follows the policy gradient,

$$\theta \leftarrow \theta + \alpha \sum_t \nabla_\theta \log \pi_\theta(s_t, a_t) A(s_t, a_t), \tag{5.2}$$

where α is the learning rate. The intuition behind the policy gradient method is as follows. The direction $\nabla_\theta \log \pi_\theta(s_t, a_t)$ specifies how to change the policy parameters in order to increase $\pi_\theta(s_t, a_t)$ (the probability of action a_t at state s_t). Equation 5.2 takes a step in this direction. The size of the step depends on the value of the advantage for action a_t in state s_t. Thus, the net effect is to reinforce actions that empirically lead to better returns.

For each experience, the advantage $A(s, a)$ is given by $R(s, a) - v^{\pi_\theta}(s)$, where $R(s, a)$ is the total reward obtained in the experience and $v^{\pi_\theta}(s)$ is the expected total reward started at state s and following the policy π_θ. The role of the *critic network* is to learn $v^{\pi_\theta}(s)$ from empirically observed rewards. We follow the standard *Temporal Difference* method [21] to train the critic network parameters θ_v,

$$\theta_v \leftarrow \theta_v - \alpha' \nabla_{\theta_v} \big(R(s) - v^{\pi_\theta}(s)\big)^2, \tag{5.3}$$

where $R(s)$ is the actual total reward sampled in the trajectory following state s, and α' is the learning rate for the critic.

For on-policy models, in order to prevent the actor and critic to over-optimize on a small portion of the environment, one common practice [20] is to add an entropy regularization term to the loss to encourage exploration, which can be critical for learning agent to converge to a good policy [31]. Concretely, we modify Equation 5.2 to be

$$\theta \leftarrow \theta + \alpha \sum_t \nabla_\theta \log \pi_\theta(s_t, a_t) A(s_t, a_t) + \beta \nabla_\theta H(\pi_\theta(\cdot|s_t)), \tag{5.4}$$

where $H(\cdot)$ takes a vector (e.g., probability distribution over actions) as input and compute the entropy. β is the balancing factor.

The detailed derivation and pseudocode can be found in [20] (§4 and Algorithm S3). It is important to note that the critic network merely helps to train the actor network. Post-training, only the actor network is required to execute the ABR algorithm and make bitrate decisions.

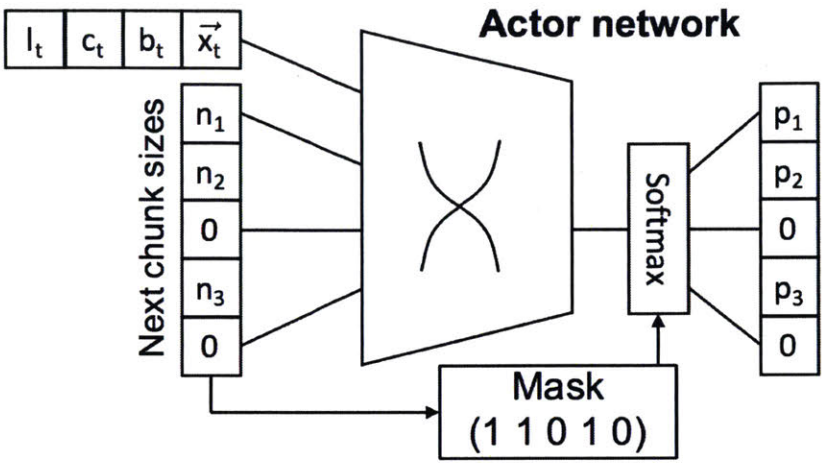


Figure 5-3: Modification to the state input and the softmax output to support multiple videos.

Parallel training: To further enhance and speed up training, Pensieve spawns multiple learning agents in parallel, as suggested by the A3C paper [20]. By default, Pensieve uses 16 parallel agents. Each learning agent is configured to experience a different set of input parameters (e.g., network traces). However, the agents continually send their {state, action, reward} tuples to a central agent, which aggregates them to generate a single ABR algorithm model. For each sequence of tuples that it receives, the central agent uses the actor-critic algorithm to compute a gradient and perform a gradient descent step. The central agent then updates the actor network and pushes out the new model to the agent which sent that tuple. Note that this can happen asynchronously among all agents, i.e., there is no locking between agents [37].

5.3 Enhancement for multiple videos

The basic algorithm described in §5.2 has some practical issues. The primary challenge is that videos can be encoded at different bitrate levels and may have diverse chunk sizes due to variable bitrate encoding [17], e.g., chunk sizes for 720p video are not identical across videos. Handling this variation would require each neural network to take a variable sized set of inputs and produce a variable sized set of outputs. Thus, the naive solution to supporting a broad range of videos is to train a model for each set of video properties that could be seen. Unfortunately, this solution is not

scalable. To overcome this, we describe two enhancements to the basic algorithm that enable Pensieve to generate a *single* model to handle multiple videos (Figure 5-3).

First, we pick canonical input and output formats that span the maximum number of bitrate levels we expect to see in practice. For example, a range of 13 levels covers the entire DASH reference client video list [38]. Then, to determine the input state for a specific video, we take the chunk sizes and map them to the index which has the closest bitrate. The remaining input states, which pertain to the bitrates that the video does not support, are zeroed out. For example, in Figure 5-3, chunk sizes (n_1, n_2, n_3) are mapped to the corresponding indices, while the remaining input values are filled with zeroes.

The second change pertains to how the output of the actor network is interpreted. For a given video, we apply a mask to the output of the final softmax layer in the actor network, such that the output probability distribution is only over the bitrates that the video actually supports. Formally, the mask is presented by a 0-1 vector $[m_1, m_2, ..., m_k]$, and the modified softmax for the NN output $[z_1, z_2, ..., z_k]$ will be

$$p_i = \frac{m_i e^{z_i}}{\sum_j m_j e^{z_j}}, \tag{5.5}$$

where p_i is the normalized probability for action i. With this modification, the output probabilities are still a continuous function of the network parameters. The reason is that the mask values $\{m_i\}$ are independent of the network parameters, and are only a function of the input video. As a result, the standard back-propagation of the gradient in the NN still holds and the training techniques established in §5.2 can be applied without modification. We evaluate the effectiveness of these modifications in more detail in §6.4.

5.4 Implementation

To generate ABR algorithms, Pensieve passes $k = 8$ past bandwidth measurements to a 1D convolution layer (CNN) with 128 filters, each of size 4 with stride 1. Next chunk sizes are passed to another 1D-CNN with the same shape. Results from these layers are then aggregated with other inputs in a hidden layer that uses 128 neurons to apply

the softmax function (Figure 5-2). The critic network uses the same NN structure, but its final output is a straight linear neuron. During training, we use a discount factor $\gamma = 0.99$, which implies that current actions will be influenced by 100 future steps. Additionally, the learning rates for actor and critic are configured to be 10^{-4} and 10^{-3}, respectively. We keep these values fixed across all our experiments. We implemented this architecture using TensorFlow [39]. For compatibility, we leveraged the TFLearn deep learning library's TensorFlow API [40] to declare the neural network during both training and testing.

Once Pensieve has generated an ABR algorithm using its simulator, it must apply the model's rules to real video streaming sessions. To do this, Pensieve runs on a standalone ABR server, implemented using the Python BaseHTTPServer. Client requests are modified to include additional information about the previous chunk download and the video being streamed (§5.2). By collecting information through client requests, Pensieve's server and ABR algorithm can remain stateless while still benefitting from observations that can solely be collected in client video players. As client requests for individual chunks arrive at the video server, Pensieve feeds all of the provided observations to its ABR algorithm. Pensieve responds to the video client with the bitrate level to use for the next chunk download; the client then contacts the appropriate CDN to fetch the corresponding chunk. It is important to note that Pensieve's ABR algorithm could also operate directly inside video players. We evaluate the overhead that a server-side deployment has on video QoE in §6.4, and discuss other deployment models in more detail in §7.

Chapter 6

Evaluation

In this section, we experimentally evaluate Pensieve. Our experiments cover a broad set of network conditions (both trace-based and in the wild) and QoE metrics. Our results answer the following questions:

1. How does Pensieve compare to state-of-the-art ABR algorithms in terms of video QoE? We find that, in all of the considered scenarios, Pensieve is able to rival or outperform the best existing scheme, with average QoE improvements ranging from 13.1%–25.0% (§6.2); Figure 6-1 provides a summary.
2. Do the models learned with Pensieve generalize to new network conditions and videos? We find that Pensieve's ABR algorithms are able to maintain high levels of performance both in the presence of new network conditions and new video properties (§6.3).
3. How sensitive is Pensieve to various system parameters such as the neural network architecture and the latency between the video client and ABR server? Our experiments suggest that performance is largely unaffected by these parameters (Tables 6.2 and 6.3). For example, applying 100 ms RTT values between clients and the Pensieve server reduces average QoE by only 3.5% (§6.4).

6.1 Methodology

Network traces: To evaluate Pensieve and state-of-the-art ABR algorithms on realistic network conditions, we created a corpus of network traces by combining several

public datasets: a broadband dataset provided by the FCC [41] and a 3G/HSDPA mobile dataset collected in Norway [42]. The FCC dataset contains over 1 million throughput traces, each of which logs the average throughput over 2100 seconds, at a 5 second granularity. We generated 1000 traces for our corpus, each with a duration of 320 seconds, by concatenating randomly selected traces from the "Web browsing" category in the August 2016 collection. The HSDPA dataset comprises 30 minutes of throughput measurements, generated using mobile devices that were streaming video while in transit (e.g., via bus, train, etc.). To match the duration of the FCC traces included in our corpus, we generated 1000 traces (each spanning 320 seconds) using a sliding window across the HSDPA dataset. To avoid scenarios where bitrate selection is trivial, i.e., situations where picking the maximum bitrate is always the optimal solution, or where the network cannot support any available bitrate for an extended period, we only considered original traces whose average throughput is less than 6 Mbps, and whose minimum throughput is above 0.2 Mbps. We reformatted throughput traces from both datasets to be compatible with the Mahimahi [43] network emulation tool. Unless otherwise noted, we used a random sample of 80% of our corpus as a training set for Pensieve; we used the remaining 20% as a test set for all ABR algorithms. All in all, our test set comprises of over 30 hours of network traces.

Adaptation algorithms: We compare Pensieve to the following algorithms which collectively represent the state-of-the-art in bitrate adaptation:

1. Buffer-Based (BB): mimics the buffer-based algorithm described by Huang et al. [16] which uses a reservoir of 5 seconds and a cushion of 10 seconds, i.e., it selects the highest bitrate that is predicted to keep the buffer occupancy above 5 seconds, and automatically chooses the highest available bitrate if the buffer occupancy exceeds 15 seconds.
2. Rate-Based (RB): predicts throughput using the harmonic mean of the experienced throughput for the past 5 chunk downloads. It then selects the highest available bitrate that is below the predicted throughput.
3. BOLA [17]: uses Lyapunov optimization to select bitrates solely considering buffer occupancy observations. We use the BOLA implementation in dash.js [23].
4. MPC [1]: uses buffer occupancy observations and throughput predictions (computed in the same way as RB) to select the bitrate which maximizes a given

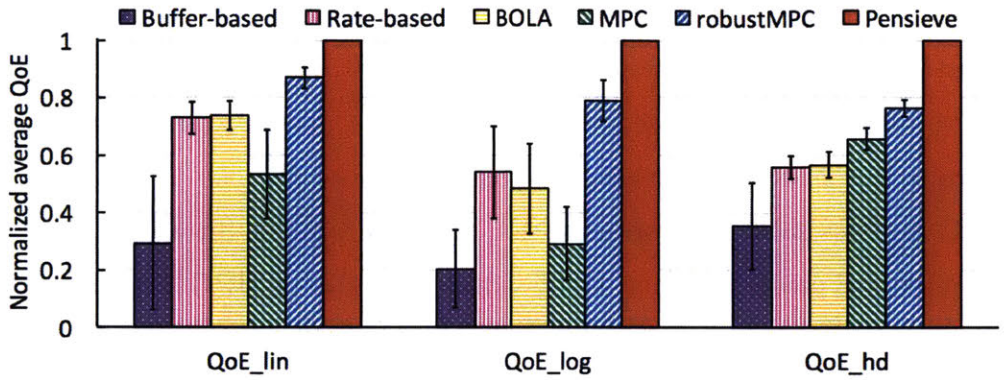
QoE metric over a horizon of 5 future chunks.

5. robustMPC [1]: uses the same approach as MPC, but accounts for errors seen between predicted and observed throughputs by normalizing throughput estimates by the max error seen in the past 5 chunks.

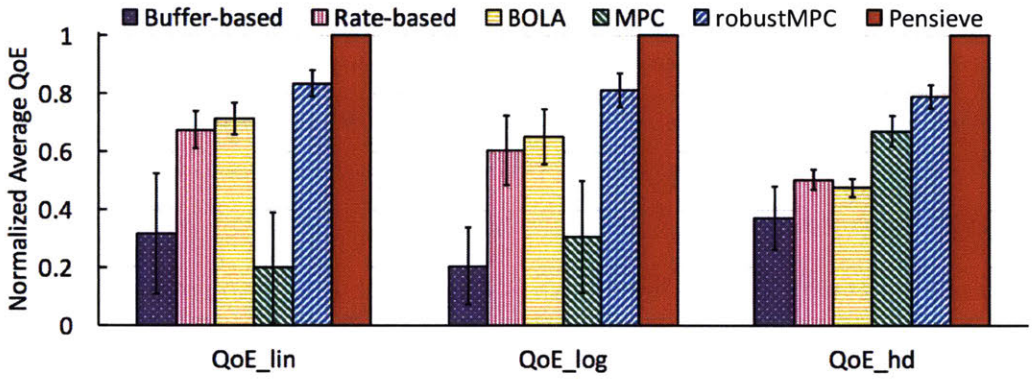
Note: MPC involves solving an optimization problem for each bitrate decision which maximizes the QoE metric over the next 5 video chunks. The MPC [1] paper describes a method, fastMPC, which precomputes the solution to this optimization problem for a quantized set of input values (e.g., buffer size, throughput prediction, etc.). Because the implementation of fastMPC is not publicly available, we implemented MPC using our ABR server as follows. For each bitrate decision, we solve the optimization problem exactly on the ABR server by enumerating all possibilities for the next 5 chunks. We found that the computation takes at most 27 ms for 6 bitrate levels and has negligible impact on QoE.

Experimental setup: We modified dash.js (version 2.4) [23] to support each of the aforementioned state-of-the-art ABR algorithms. For Pensieve and both variants of MPC, dash.js was configured to fetch bitrate selection decisions from an ABR server that implemented the corresponding algorithm. ABR servers ran on the same machine as the client, and requests to these servers were made using `XMLHttpRequests`. All other algorithms ran directly in dash.js. The DASH player was configured to have a playback buffer capacity of 60 seconds. Our evaluations used the "EnvivioDash3" video from the DASH-246 JavaScript reference client [38]. This video is encoded by the H.264/MPEG-4 codec at bitrates in $\{300, 750, 1200, 1850, 2850, 4300\}$ kbps (which pertain to video modes in $\{240, 360, 480, 720, 1080, 1440\}$p). Additionally, the video was divided into 48 chunks and had a total length of 193 seconds. Thus, each chunk represented approximately 4 seconds of video playback. In our setup, the client video player was a Google Chrome browser (version 53) and the video server (Apache version 2.4.7) ran on the same machine as the client. We used Mahimahi [43] to emulate the network conditions from our corpus of network traces, along with an 80 ms RTT, between the client and server. Unless otherwise noted, all experiments were performed on Amazon EC2 t2.2xlarge instances.

QoE metrics: There exists significant variance in user preferences for video streaming QoE [11, 12, 25, 13]. Thus, we consider a variety of QoE metrics. We start with



(a) FCC broadband dataset



(b) Norway HSDPA dataset

Figure 6-1: Comparing Pensieve with existing ABR algorithms on broadband and 3G/HSDPA networks. The QoE metrics considered are presented in Table 6.1. Results are normalized against the performance of Pensieve. Error bars span ± one standard deviation from the average.

the general QoE metric used by MPC [1], which is defined as

$$QoE = \sum_{n=1}^{N} q(R_n) - \mu \sum_{n=1}^{N} T_n - \sum_{n=1}^{N-1} \left| q(R_{n+1}) - q(R_n) \right| \tag{6.1}$$

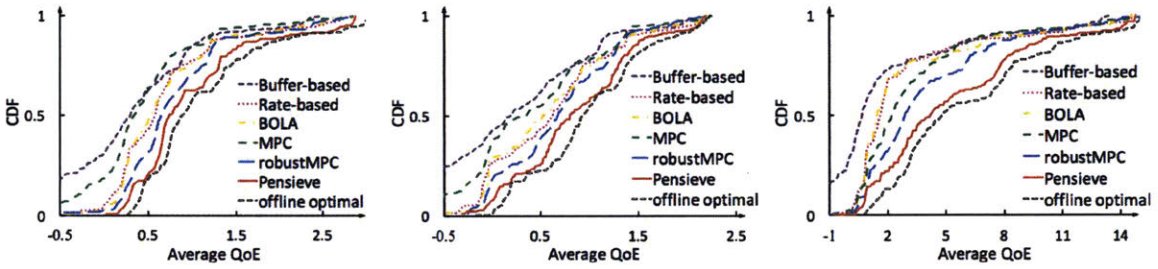
for a video with N chunks. R_n represents the bitrate of $chunk_n$ and $q(R_n)$ maps that bitrate to the quality perceived by a user. T_n represents the rebuffering time that results from downloading $chunk_n$ at bitrate R_n, while the final term penalizes changes in video quality to favor smoothness.

We consider three choices of $q(R_n)$:

1. QoE_{lin}: $q(R_n) = R_n$. This metric was used by MPC [1].
2. QoE_{log}: $q(R_n) = \log(R/R_{min})$. This metric captures the notion that, for some

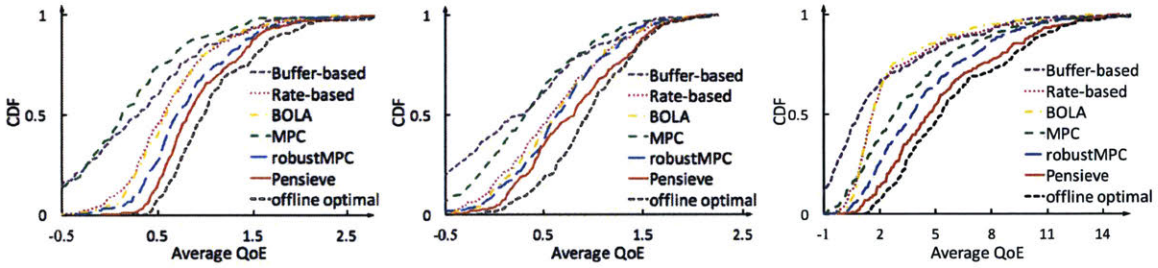
Name	bitrate utility ($q(R)$)	rebuffer penalty (μ)
QoE_{lin}	R	4.3
QoE_{log}	$\log(R/R_{min})$	2.66
QoE_{hd}	0.3→1, 0.75→2, 1.2→3 1.85→12, 2.85→15, 4.3→20	8

Table 6.1: The QoE metrics we consider in our evaluation. Each metric is a variant of Equation 6.1.



(a) QoE_{lin} (b) QoE_{log} (c) QoE_{hd}

Figure 6-2: Comparing Pensieve with existing ABR algorithms on the QoE metrics listed in Table 6.1. Results were collected on the FCC broadband dataset. Average QoE values are listed for each ABR algorithm.



(a) QoE_{lin} (b) QoE_{log} (c) QoE_{hd}

Figure 6-3: Comparing Pensieve with existing ABR algorithms on the QoE metrics listed in Table 6.1. Results were collected on the Norway HSDPA dataset. Average QoE values are listed for each ABR algorithm.

users, the marginal improvement in perceived quality decreases at higher bitrates. This metric was used by BOLA [17].

3. QoE_{hd}: This metric favors High Definition (HD) video. It assigns a low quality score to non-HD bitrates and a high quality score to HD bitrates.

The exact values of $q(R_n)$ for our baseline video are provided in Table 6.1. In this section, we report the average QoE per chunk, i.e., the total QoE metric divided by

the number of chunks in the video.

6.2 Pensieve vs. Existing ABR algorithms

To evaluate Pensieve, we compared it with state-of-the-art ABR algorithms on each QoE metric listed in Table 6.1. In each experiment, Pensieve's ABR algorithm was trained to optimize for the considered QoE metric, using the entire training corpus described in §6.1; both MPC variants were also modified to optimize for the considered QoE metric. Figure 6-1 shows the average QoE that each scheme achieves on our entire test corpus. Figures 6-2 and 6-3 provide more detailed results in the form of full CDFs for each network. As a comparison, we compute the offline[1] optimal using dynamic programing with future throughput information.

There are two key takeaways from these results. First, we find that Pensieve either matches or exceeds the performance of the best existing ABR algorithm on each QoE metric and network considered. The closest competing scheme is robustMPC; this shows the importance of tuning, as without robustMPC's conservative throughput estimates, MPC can become too aggressive (relying on the playback buffer) and perform worse than even a naive rate-based scheme. For QoE_{lin}, which was considered in the MPC paper [1], the average QoE for Pensieve is 13.1% higher than robustMPC on the FCC broadband network traces. The gap between Pensieve and robustMPC widens to 18.5% and 30.4% for QoE_{log} and QoE_{hd}. The results are qualitatively similar for the Norway HSDPA network traces.

Second, we observe that the performance of existing ABR algorithms is sensitive to different QoE objectives. The reason is that these algorithms employ fixed control laws, even though optimizing for different QoE objectives requires inherently different ABR strategies. For example, unlike QoE_{lin}, the optimal strategy for QoE_{log} is to make small increases in bitrate since the marginal improvement in user-perceived quality diminishes at higher bitrates. With this strategy, video players avoid jumping to high bitrate levels when the risk of rebuffering is high. However, to optimize for QoE_{lin}, the ABR algorithm needs to be more aggressive. Pensieve is able to

[1]Notice that the offline optimal is not realistic as it has full knowledge of the future. It only serves as an upper bound of QoE obtained by any possible sequence of decisions. In §6.4, we perform detailed analysis of the practical optimality gap with an online optimal scheme.

automatically learn these policies (without explicit tuning) and thus, performance with Pensieve remains consistently high as conditions change.

The results for QoE_{hd} further illustrate this point. Recall that QoE_{hd} favors HD video, assigning the highest utility to the top three bitrates available for our test video (see Table 6.1). As discussed in §4, optimizing for QoE_{hd} requires significantly more long-term planning than the other two QoE metrics. When network bandwidth is inadequate, the ABR algorithm should build the playback buffer as quickly as possible using the lowest available bitrate. Once the buffer is large enough, it should then make a direct transition to the lowest HD quality (bypassing intermediate bitrates). However, building buffers to a level which circumvents rebuffering and maintains sufficient smoothness requires a lot of foresight. As illustrated by the example in Figure 4-2b, Pensieve is able to learn such a policy, while robustMPC's conservative throughput predictions and 5 chunk horizon prevent it from doing so. It may be possible to tune robustMPC to better cater to QoE_{hd}, e.g., by increasing the horizon length and reducing the conservatism of the throughput predictor. However, such tweaks may not perform well on other QoE metrics. In contrast, Pensieve learns a good ABR policy purely from experience, with zero tuning or designer involvement.

6.3 Generalization

In the experiments above, Pensieve was trained with a set of traces collected on the same networks that were used during testing; note that no test traces were directly included in the training set. However, in practice, Pensieve's ABR algorithms could encounter new networks, with different conditions (and thus, with different optimal strategies). To evaluate Pensieve's ability to generalize to new network conditions, we conduct two experiments. First, we evaluate Pensieve in the wild on two real networks. Second, we show how Pensieve can be trained to perform well across multiple environments using a purely synthetic dataset.

Real world experiments: We evaluated Pensieve and several state-of-the-art ABR algorithms in the wild using two networks: a public WiFi network at a local coffee shop, and the Verizon LTE cellular network. In these experiments, a client, running on a Macbook Pro laptop, contacted a video server running on a nearby desktop

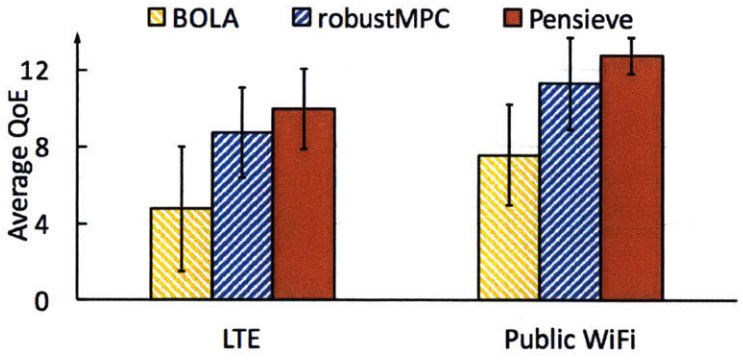
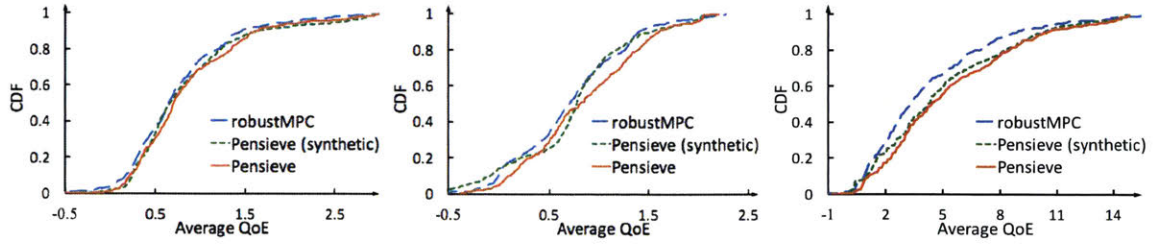


Figure 6-4: Comparing Pensieve with existing ABR algorithms in the wild on the QoE_{hd} metric. Results were collected using a public WiFi network and the Verizon LTE cellular network. Bars list averages and error bars span ± one standard deviation from the average.

machine. We considered a subset of the ABR algorithms listed in §6.1: BOLA, robustMPC, and Pensieve. On each network, we loaded our test video five times with each scheme, randomly selecting the order among them. The Pensieve ABR algorithm evaluated here was solely trained using the broadband and HSDPA traces in our corpus. However, even on these new networks, Pensieve was able to outperform the existing ABR algorithms on the QoE_{hd} metric (Figure 6-4). The reason is the same as described above: because existing metrics were not manually tuned for QoE_{hd}, they fail to plan far enough into the future, requesting chunks at bitrates just below HD quality. In contrast, Pensieve's ABR algorithm automatically learned to generalize the strategy it developed for QoE_{hd} on the training networks to the new networks seen in the wild.



(a) QoE_{lin} (b) QoE_{log} (c) QoE_{hd}

Figure 6-5: Comparing two ABR algorithms with Pensieve on the broadband and HSDPA networks: one algorithm was trained on synthetic network traces, while the other was trained using a set of traces directly from the broadband and HSDPA networks. Results are aggregated across the two datasets.

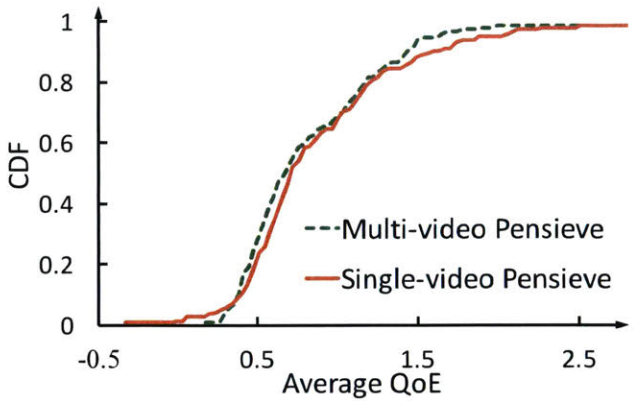


Figure 6-6: Comparing ABR algorithms trained across multiple videos with those trained explicitly on the test video. The measuring metric is QoE_{lin}.

Training with a synthetic dataset: Can we train Pensieve without *any* real network data? Learning from synthetic data alone would of course be undesirable, but we use it as a challenging test of Pensieve's ability to generalize.

We design a data set to cover a relatively broad set of network conditions, with average throughputs ranging from 0.2 Mbps to 4.3 Mbps. Specifically, the dataset was generated using a Markovian model in which each state represented an average throughput in the aforementioned range. State transitions were performed at a 1 second granularity and followed a geometric distribution (making it more likely to transition to a nearby average throughput). Each throughput value was then drawn from a Gaussian distribution centered around the average throughput for the current state, with variance uniformly distributed between 0.05 and 0.5.

We then used Pensieve to compare two ABR algorithms on the test dataset described above (i.e., a combination of the HSDPA and broadband datasets): one trained solely using the synthetic dataset, and another trained explicitly on broadband and HSDPA network traces. Figure 6-5 illustrates our results for all three QoE metrics listed in Table 6.1. As shown, Pensieve's ABR algorithm that was trained on the synthetic dataset is able to generalize across these new networks, outperforming robustMPC and achieving average QoE values within 1.4%–11.9% of the ABR algorithm trained directly on the test networks. These results suggest that, in practice, Pensieve will likely be able to generalize to a broad range of network conditions encountered by its clients.

Multiple videos: As a final test of generalization, we evaluated Pensieve's ability to generalize across multiple video properties. To do this, we trained a single ABR algorithm on 1,000 synthetic videos using the techniques described in §5.3. Specifically, the number of bitrate is randomly selected from $[3, 10]$ levels. The bitrates are then randomly chosen from $\{200, 300, 450, 750, 1200, 1850, 2350, 2850, 3500, 4300\}$ kbps. The number of video chunks is randomly generated from $[20, 100]$ chunks and the actual file size of each 4-second video chunk is multiplied with a Gaussian noise $\sim \mathcal{N}(0, 0.1)$ to synthesize the variation of file size. Thus, these videos diverge on numerous properties including the bitrate options (both the number of options and value of each), number of chunks, chunk sizes and video duration. Additionally, none of the generated training videos overlaps the testing video on the bitrates. Unsurprisingly, the number of available bitrates for these videos represent the two ends of the spectrum for videos provided by the DASH reference client [38].

We compare this newly trained model to the original model, which is trained solely on the "EnvivioDash3" video described in §6.1. Our results measure QoE_{lin} on broadband and HSDPA network traces and are depicted in Figure 6-6. As shown, the generalized ABR algorithm from multi-video model is able to achieve average QoE_{lin} values within 3.22% of models trained explicitly on the test video. These results suggest that in practice, Pensieve servers can be configured to use a small number of ABR algorithms to improve streaming for a diverse set of videos.

6.4 Pensieve Deep Dive

Pensieve's default implementation raises three practical concerns. How sensitive is Pensieve to the structure of its learning architecture? What is the overhead of training ABR algorithms and using the resulting models to guide client chunk downloads? What impact does the additional latency incurred by clients to retrieve bitrate suggestions from Pensieve's video servers have on client-perceived QoE? In this section, we describe fine-grained experiments that shed light on these challenges and explain the feasibility of using RL-generated ABR algorithms in real video streaming sessions. All experiments in this section used the experimental setup described in §6.1 and consider the QoE_{hd} metric.

Number of neurons and filters (each)	Average QoE_{hd}
4	3.850 ± 1.215
16	4.681 ± 1.369
32	5.106 ± 1.452
64	5.496 ± 1.411
128	5.489 ± 1.378

Table 6.2: Sweeping the number of CNN filters and hidden neurons in Pensieve's learning architecture.

Number of hidden layers	Average QoE_{hd}
1	5.489 ± 1.378
2	5.396 ± 1.434
5	4.253 ± 1.219

Table 6.3: Sweeping the number of hidden layers in Pensieve's learning architecture.

Neural Network (NN) architecture: Starting with Pensieve's default learning architecture (Figure 5-2), we swept a range of NN parameters to understand the impact that each has on user-perceived QoE. First, using a fixed single hidden layer, we varied the number of filters in the 1D-CNN and the number of neurons in the hidden merge layer. These parameters were swept in tandem, i.e., when 4 filters were used, 4 neurons were used. Results from this sweep are presented in Table 6.2. As shown, performance begins to plateau once the number of filters and neurons each exceed 32. Additionally, notice that once these values reach 128 (Pensieve's default configuration), variance levels decrease while average QoE_{hd} values remain stable.

Next, after fixing the number of filters and hidden neurons to 128, we varied the number of hidden layers in Pensieve's architecture. The resulting QoE_{hd} values are listed in Table 6.3. Interestingly, we find that the shallowest network of 1 hidden layer yields the best performance; this represents the default value in Pensieve. Performance steadily degrades as we increase the number of hidden layers. However, it is important to note that our sweep used a fixed learning rate and number of training iterations. Tuning these parameters to cater to deeper networks may improve performance, as these networks generally take longer to train.

Training time: To measure the overhead of generating ABR algorithms using RL, we profiled Pensieve's training process. Training a single algorithm required approximately 50,000 iterations, where each iteration took 300 ms and corresponded to 16 agents updating their parameters in parallel (using the asynchronous training ap-

RTT (ms)	Average QoE_{hd}
0	5.407 ± 1.820
20	5.356 ± 1.768
40	5.309 ± 1.768
60	5.271 ± 1.773
80	5.217 ± 1.742
100	5.219 ± 1.748

Table 6.4: Average QoE_{hd} values when different RTT values are imposed between the client and Pensieve server.

proach described in §5.2). Thus, in total, training took approximately 4 hours. We note that this cost is incurred offline and can be performed infrequently depending on environment stability.

Client-to-ABR server latency: Recall that with Pensieve, RL-generated ABR algorithms are applied to video streaming sessions by servers (not clients). Under this deployment model, clients must first query the Pensieve server to determine the bitrate to use for the next chunk, before downloading that chunk from a CDN server. To understand the overhead incurred by this additional round trip, we performed a sweep of the RTT between the client player and Pensieve server, considering values from 0 ms–100 ms. This experiment used the same setup described in §6.1, and measured the QoE_{hd} metric. Table 6.4 lists our results, highlighting that the latency from this additional RTT has minimal impact on QoE: the average QoE_{hd} with a 100 ms latency was within 3.5% of that when the latency was 0 ms. The reason is that the latency incurred from the additional round trip to Pensieve's server is masked by the playback buffer occupancy and chunk download times [6, 15].

Online and offline optimality: How is the performance of Pensieve compared with an optimal scheme? Notice that there still remains a sizable gap between Pensieve and *offline* optimal as shown in Figure 6-3 and 6-2. However, the offline optimal in §6.1 is obtained by running dynamic programming with the omniscient knowledge of the future bandwidth. To reflect the realistic *online* optimal, it requires to know the underlining distribution of the future network throughput. Therefore, we conduct a controlled experiment where the chunk download time is generated following an

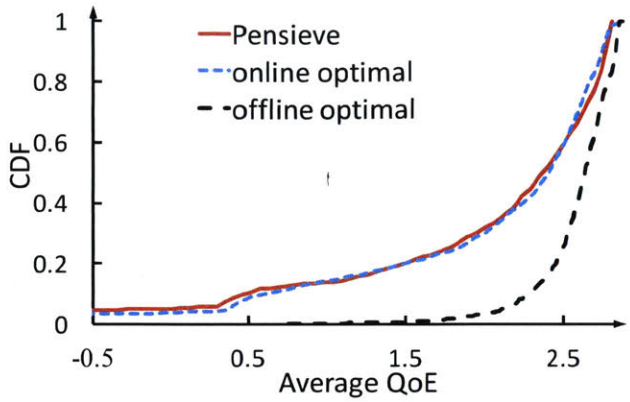


Figure 6-7: Comparing Pensieve with online and offline optimal with QoE_{lin} metric.

known Markov process. Specifically, we simulate the download time T_n of chunk n as

$$T_n = T_{n-1}\frac{R_n}{R_{n-1}} + \epsilon, \tag{6.2}$$

where R_n is the bitrate of chunk n and ϵ is an additive Gaussian noise.

We can then write down the dynamic programming procedure for finding the online optimal decision,

$$QoE_n(B_n, T_n, R_n) = \max_{R_{n+1}} \Bigg\{ q(R_{n+1}) - \mu \left[T_n \frac{R_{n+1}}{R_n} - B_n \right]_+ - \left| q(R_{n+1}) - q(R_n) \right| + \mathbb{E}_{T_{n+1}} \left[QoE_{n+1}(B_{n+1}, T_{n+1}, R_{n+1}) \right] \Bigg\}, \tag{6.3}$$

$$B_{n+1} = \left[B_n - T_{n+1} \right]_+ + \delta, \tag{6.4}$$

$$T_{n+1} \sim \mathcal{N}\left(T_n \frac{R_{n+1}}{R_n}, \sigma^2 \right), \tag{6.5}$$

where B_n is the buffer occupancy right after chunk n is downloaded, which depends on the chunk download time T_n and the size of chunk δ. The chunk download time T_n yields a Gaussian distribution.

In our experiment, the evaluation metric follows QoE_{lin} in Table 6.1. The video chunk length δ is 4 seconds, using the setting of "EnvivioDash3" video described in §6.1. To generate network traces with similar range of the video bitrates, the ini-

tial download time T_0 is set to 4 seconds for bitrate $R_0 = 2$kbps, and the standard deivation σ of the additive Guassian noise is configured to be 0.5. Additionally, Equations 6.4 and 6.5 are confined in $[0, 30]$ seconds to avoid unrealistic chunk download behavior (e.g., negative download time). The granularity of dynamic programming is 0.1 seconds for both buffer occupancy and download time.

We use the same setup in §6.1 to train a Pensieve agent in this simulated environment, and compare the performance with online and offline optimal. The results are depicted in Figure 6-7. Recall that the offline dynamic programing uses the *exact* future download time, whereas the online one only uses the corresponding *distribution*. Therefore, as expected, the offline optimal outperforms the online optimal by 9.1% on average, which is in the similar scale as the optimality gap observed in §6.2. Meanwhile, notice that the average QoE achieved by Pensieve is within 0.2% of the online optimal. This near-optimal performance implies that Pensieve is able to learn the underlining distribution of download time through experience, and it can learn an optimal online policy by interacting with the video streaming environment directly.

Chapter 7

Discussion

Deploying Pensieve in practice: In our current implementation, Pensieve's ABR server runs on the server-side of video streaming applications. This approach offers several advantages over deployment in client video players. First, a variety of client-side devices are used for video streaming today, ranging from multi-core desktop machines to mobile devices to TVs. By using an ABR server to simply guide client bitrate selection, Pensieve can easily support this broad range of video clients without modifications that may sacrifice performance. Additionally, ABR algorithms are traditionally deployed on clients who can quickly react to changing environments [1]. However, as noted in §5, Pensieve preserves this ability by having clients include observations about the environment in each request sent to the ABR server. Further, our results suggest that the additional latency required to contact Pensieve's ABR server has negligible impact on QoE (§6.4). If direct deployment in client video players is preferred, Pensieve could use compressed neural networks [44] or represent them in languages supported by many client applications, e.g., JavaScript [45].

Online training: In this thesis, we primarily described RL-based ABR algorithm generation as an offline task. That is, with Pensieve, we assumed that the ABR algorithm was generated a priori (during a training phase) and was then unmodified after deployment. However, Pensieve can naturally support an *online* approach in which an ABR algorithm is generated or updated based on observations made by its clients. This technique would enable ABR algorithms to further adapt to the exact conditions being experienced at a given time. However, several factors must be

considered with this approach. First, direct deployment of ABR algorithms in client video players (as described above) is more challenging with the online approach as the computation footprint is increased due to intermittent training. Second, conventional neural networks favor recent observations, often forgetting past conditions unless they are explicitly stored and trained with. Thus, using an online training approach could lead to bias towards short-lived network conditions. To overcome this, Pensieve's learning architecture could be modified to support other networks like recurrent neural networks (RNNs) and Long short-term memory (LSTM) networks which are designed to remember larger amounts of history.

Chapter 8

Conclusion

We presented Pensieve, a system which generates ABR algorithms using Reinforcement Learning. Unlike existing ABR algorithms that use fixed heuristics which have been tuned to specific environments, Pensieve's ABR algorithms are generated solely using observations of the resulting performance of past decisions. By doing this, Pensieve is able to overcome many of the challenges that existing algorithms face, including the inability to generalize to different environments. Over a broad set of network conditions and QoE metrics, we found that Pensieve was able to outperform existing ABR algorithms by 13.1%–25.0%.

Bibliography

[1] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. A Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM. ACM, 2015.

[2] Cisco. Cisco visual networking index: Forecast and methodology, 2015-2020. 2016.

[3] Sandvine. Global internet phenomena-latin american & north america. 2015.

[4] S. Shunmuga Krishnan and Ramesh K. Sitaraman. Video Stream Quality Impacts Viewer Behavior: Inferring Causality Using Quasi-experimental Designs. In *Proceedings of the 2012 ACM Conference on Internet Measurement Conference*, IMC. ACM, 2012.

[5] Florin Dobrian, Vyas Sekar, Asad Awan, Ion Stoica, Dilip Joseph, Aditya Ganjam, Jibin Zhan, and Hui Zhang. Understanding the Impact of Video Quality on User Engagement. In *Proceedings of the ACM SIGCOMM 2011 Conference*, SIGCOMM. ACM, 2011.

[6] Te-Yuan Huang, Nikhil Handigol, Brandon Heller, Nick McKeown, and Ramesh Johari. Confused, Timid, and Unstable: Picking a Video Streaming Rate is Hard. In *Proceedings of the 2012 ACM Conference on Internet Measurement Conference*, IMC. ACM, 2012.

[7] Yi Sun, Xiaoqi Yin, Junchen Jiang, Vyas Sekar, Fuyuan Lin, Nanshu Wang, Tao Liu, and Bruno Sinopoli. CS2P: Improving Video Bitrate Selection and Adaptation with Data-Driven Throughput Prediction. In *Proceedings of the 2016 Conference on ACM SIGCOMM 2016 Conference*, SIGCOMM. ACM, 2016.

[8] Xuan Kelvin Zou, Jeffrey Erman, Vijay Gopalakrishnan, Emir Halepovic, Rittwik Jana, Xin Jin, Jennifer Rexford, and Rakesh K. Sinha. Can accurate predictions improve video streaming in cellular networks? In *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications*, HotMobile. ACM, 2015.

[9] Keith Winstein, Anirudh Sivaraman, and Hari Balakrishnan. Stochastic Forecasts Achieve High Throughput and Low Delay over Cellular Networks. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, NSDI'13, Berkeley, CA, USA. USENIX Association.

[10] Yasir Zaki, Thomas Pötsch, Jay Chen, Lakshminarayanan Subramanian, and Carmelita Görg. Adaptive congestion control for unpredictable cellular networks. In *ACM SIGCOMM Computer Communication Review*, volume 45, pages 509–522. ACM, 2015.

[11] István Ketykó, Katrien De Moor, Toon De Pessemier, Adrián Juan Verdejo, Kris Vanhecke, Wout Joseph, Luc Martens, and Lieven De Marez. QoE Measurement of Mobile YouTube Video Streaming. In *Proceedings of the 3rd Workshop on Mobile Video Delivery*, MoViD. ACM, 2010.

[12] Ricky K.P. Mok, Edmond W.W. Chan, Xiapu Luo, and Rocky K.C. Chang. Inferring the QoE of HTTP Video Streaming from User-viewing Activities. In *Proceedings of the First ACM SIGCOMM Workshop on Measurements Up the Stack*, W-MUST. ACM, 2011.

[13] Kandaraj Piamrat, Cesar Viho, Jean-Marie Bonnin, and Adlen Ksentini. Quality of Experience Measurements for Video Streaming over Wireless Networks. In *Proceedings of the 2009 Sixth International Conference on Information Technology: New Generations*, ITNG. IEEE Computer Society, 2009.

[14] Iraj Sodagar. The mpeg-dash standard for multimedia streaming over the internet. *IEEE MultiMedia*, 18(4):62–67, 2011.

[15] Junchen Jiang, Vyas Sekar, and Hui Zhang. Improving Fairness, Efficiency, and Stability in HTTP-based Adaptive Video Streaming with FESTIVE. In *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, CoNEXT. ACM, 2012.

[16] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. A Buffer-based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM. ACM, 2014.

[17] Kevin Spiteri, Rahul Urgaonkar, and Ramesh K. Sitaraman. BOLA: near-optimal bitrate adaptation for online videos. volume abs/1601.06748, 2016.

[18] 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. volume 32, pages 719–733, April 2014.

[19] Joanne K Rowling. Harry potter and the goblet of fire. *London: Bloomsbury*, 2000.

[20] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *CoRR*, 2016.

[21] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[22] Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula. Resource management with deep reinforcement learning. In *HotNets*. ACM, 2016.

[23] Akamai. dash.js. https://github.com/Dash-Industry-Forum/dash.js/, 2016.

[24] Saamer Akhshabi, Ali C. Begen, and Constantine Dovrolis. An experimental evaluation of rate-adaptation algorithms in adaptive streaming over http. In *Proceedings of the Second Annual ACM Conference on Multimedia Systems*, MMSys. ACM, 2011.

[25] R. K. P. Mok, E. W. W. Chan, and R. K. C. Chang. Measuring the quality of experience of HTTP video streaming. In *12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops*, May 2011.

[26] Junchen Jiang, Vyas Sekar, Henry Milner, Davis Shepherd, Ion Stoica, and Hui Zhang. CFA: A Practical Prediction System for Video QoE Optimization. In *Proceedings of the 13th Usenix Conference on Networked Systems Design and Implementation*, NSDI. USENIX Association, 2016.

[27] Federico Chiariotti. Reinforcement learning algorithms for dash video streaming. In *Department of Information Engineering, University of Padova*, 2015.

[28] Federico Chiariotti, Stefano D'Aronco, Laura Toni, and Pascal Frossard. Online learning adaptation strategy for dash clients. In *Proceedings of the 7th International Conference on Multimedia Systems*, page 8. ACM, 2016.

[29] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, Demis Hassabis Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Human-level control through deep reinforcement learning. *Nature*, 2015.

[30] David Silver, Aja Huang, Christopher J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529:484–503, 2016.

[31] Yuxin Wu and Yuandong Tian. Training agent for first-person shooter game with actor-critic curriculum learning. In *Submitted to Intl Conference on Learning Representations*, 2017.

[32] Mark Allman, Vern Paxson, and Ethan Blanton. Tcp congestion control. *RFC 5681*, 2009.

[33] G Fairhurst, A Sathiaseelan, and R Secchi. Updating tcp to support rate-limited traffic. *RFC 7661*, 2015.

[34] M Handley, J Padhye, and S Floyd. Tcp congestion window validation. *RFC 2861*, 2000.

[35] Martin T Hagan, Howard B Demuth, Mark H Beale, and Orlando De Jesús. *Neural network design*. PWS publishing company Boston, 1996.

[36] Richard S Sutton, David A McAllester, Satinder P Singh, Yishay Mansour, et al. Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, volume 99, pages 1057–1063, 1999.

[37] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems*, pages 693–701, 2011.

[38] DASH Industry Form. Reference Client 2.4.0. http://mediapm.edgesuite.net/dash/public/nightly/samples/dash-if-reference-player/index.html, 2016.

[39] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: A System for Large-scale Machine Learning. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI. USENIX Association, 2016.

[40] TFLearn. TFLearn: Deep learning library featuring a higher-level API for TensorFlow. http://tflearn.org/, 2017.

[41] Federal Communications Commission. Raw Data - Measuring Broadband America 2016. https://www.fcc.gov/reports-research/reports/measuring-broadband-america/raw-data-measuring-broadband-america-2016, 2016.

[42] Haakon Riiser, Paul Vigmostad, Carsten Griwodz, and Pål Halvorsen. Commute Path Bandwidth Traces from 3G Networks: Analysis and Applications. In *Proceedings of the 4th ACM Multimedia Systems Conference*, MMSys. ACM, 2013.

[43] Ravi Netravali, Anirudh Sivaraman, Somak Das, Ameesh Goyal, Keith Winstein, James Mickens, and Hari Balakrishnan. Mahimahi: Accurate Record-and-Replay for HTTP. In *Proceedings of USENIX ATC*, 2015.

[44] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *CoRR, abs/1510.00149*, 2, 2015.

[45] Synaptic. synaptic.js – The javascript architecture-free neural network library for node.js and the browser. https://synaptic.juancazala.com/, 2016.