Continual learning improves Internet video streaming

Francis Y. Yan, Hudson Ayers, Chenzhi Zhu†, Sadjad Fouladi,
James Hong, Keyi Zhang, Philip Levis, and Keith Winstein
Stanford University, † Tsinghua University

ABSTRACT

We describe Fugu, a *continual learning* algorithm for bitrate selection in streaming video. Each day, Fugu retrains a neural network from its experience in deployment over the prior week. The neural network predicts how long it would take to transfer each available version of the upcoming video chunks, given recent history and internal TCP statistics.

We evaluate Fugu with Puffer[1], a public website we built that streams live TV using Fugu and existing algorithms. Over a nine-day period in January 2019, Puffer streamed 8,131 hours of video to 3,719 unique users. Compared with buffer-based control, MPC, RobustMPC, and Pensieve, Fugu performs better on several metrics: it stalls 5–13× less and has better and more stable video quality, and users who were randomly assigned to Fugu streamed for longer on average before quitting or reloading. We find that TCP statistics can aid bitrate selection, that congestion-control and bitrate-selection algorithms are best selected jointly, that probabilistic transfer-time estimates that consider chunk size outperform point estimates of throughput, and that both stalls and video quality have an influence on how long users choose to keep a video stream playing.

Fugu's results suggest that continual learning of network algorithms *in situ* is a promising area of research. To support further investigation, we plan to operate Puffer for several years and will open it to researchers for evaluating new congestion-control and bitrate-selection approaches.

1 INTRODUCTION

Video streaming is the predominant Internet application, accounting for almost three quarters of the Internet's traffic [30]. Considerable research has focused on this area, especially the problem of *adaptive bitrate selection*, or ABR: choosing which compression level to download for each "chunk," or segment, of the video. ABR algorithms generally try to optimize quality of experience (QoE), striving for high, stable video quality while avoiding stalls caused by underflow of the client's playback buffer.

Because ABR depends on many variables, including buffer occupancy, throughput, chunk size, and delay, many modern ABR algorithms use statistical and machine-learning methods. MPC/RobustMPC [33] uses techniques from robotics and operations research to plan an optimal sequence of upcoming chunks to download. BOLA [26, 27] models the connection's varying throughput as a continuous stationary random process. CS2P [29] models the throughput as drawn from a finite-state process and trains a hidden Markov model to detect state changes and estimate the most likely throughput. Oboe [2] detects when a connection appears to change state, then adjusts the parameters of an ABR algorithm to optimize QoE. Pensieve [15] teaches a neural network to perform video bitrate selection without an explicit model of the problem; the system was trained in simulation and rewarded for decisions that produced good QoE.

The use of machine learning arises naturally in this context; ML brings powerful tools that can generate good algorithms from a sea of conflicting data. Machine learning has an Achilles' heel, however. If the training and test environments differ, the generated algorithm may perform poorly. This phenomenon may be due to model mismatch, e.g., when the learned algorithm is overfit to the training set, or to dataset shift, when the test set changes over time [22].

Unfortunately, learned networking algorithms can be especially vulnerable to these problems. The Internet is complex and diverse, so algorithms trained in emulation or on small datasets may fail to anticipate conditions that arise in deployment. The Internet is also dynamic: algorithms trained on conditions or users from a month ago may be out of date. These effects make it harder for good performance in simulation or training to translate to the Internet. For example, CS2P's gains were more modest over real network paths than in simulation [29]. An attempt to reproduce Pensieve's performance was not able to replicate its gains when running over similar, but not identical, network paths [8]. Other learned algorithms, such as the Remy congestion-control schemes, have also seen inconsistent performance on real networks [32]. The brittleness of some learned algorithms has led to questions about when they can safely be deployed on networks where no participant has complete information and failure may not be immediately detectable [28].

In this paper, we present Fugu, a *continual learning* algorithm for bitrate selection in streaming video. Fugu mitigates the risk of model mismatch by learning in the same environment where it is deployed. To counter dataset shift, Fugu learns continuously from its users, producing testable predictions whose accuracy can be monitored. Fugu's key innovations lie in explicitly considering the dynamic nature of today's networks at several time scales. At the time scale of

[1] https://puffer.stanford.edu

Algorithm (over BBR congestion control)	Time stalled (lower is better)	Mean SSIM (higher is better)	SSIM variation (mean, lower is better)
Fugu	0.06%	17.03 dB	0.53 dB
Buffer-based [11]	0.34%	16.58	0.79
MPC [33]	0.81%	16.32	0.54
RobustMPC [33]	0.29%	15.74	0.63
Pensieve [15]	0.52%	16.19	0.88
Non-continual Fugu	0.31%	16.49	0.60
Emulation-trained Fugu	0.76%	15.26	1.03

Figure 1: Results from January 19–28, 2019, showing BBR [6] streams (3,470 hours in total). Fugu outperformed the other schemes, including versions of Fugu trained non-continually on Puffer or in emulation.

days and weeks, Fugu adapts its behavior based on observed experience. At the time scale of seconds and minutes, Fugu replaces the instantaneous *throughput predictor* of previous schemes with a *probabilistic transmission-time predictor* of proposed chunks. A transmission-time predictor can capture the fact that throughput may vary over the duration of a chunk's transmission. It is structured as a neural network trained daily to make fuzzy predictions as a function of available data, including lower-layer congestion-control statistics. We found that several of Fugu's contributions quantitatively improve the QoE of Internet video streaming:

- training in the same environment as deployment,
- retraining daily from experience over the prior week,
- predicting "how long would it take to transmit a chunk of length x at time t?" instead of estimating throughput,
- making fuzzy predictions instead of point estimates,
- combining model-based control and a neural network (known as model-based reinforcement learning), and
- incorporating information not previously used in ABR, such as internal TCP and congestion-control statistics.

To evaluate Fugu and other algorithms, we built Puffer, a public, live TV-streaming website. Puffer is a working testbed to experiment with ABR and congestion-control algorithms to a broad set of users. Puffer receives six broadcast television channels and encodes them for streaming, served from one server in a well-connected datacenter with a 10 Gbps uplink. New video streams are randomly assigned to a congestion-control and ABR scheme. Users are blinded to the assignment.

Fugu's results suggest that continual learning of ABR algorithms *in situ* provides a substantial benefit to real users on real networks. In 8,131 hours of video streamed by 3,719 people from January 19–28, 2019, Fugu outperformed buffer-based control [11], MPC, RobustMPC, and Pensieve. It also outperformed an emulator-trained version of Fugu, and a version of Fugu statically trained *once* on real-world data from Puffer. Fugu had the lowest rebuffering time (by margins of 5–13×), the greatest average video quality (measured with structural similarity, or SSIM [31]), and the smallest chunk-to-chunk variation in quality. Figure 1 shows a summary of major results; complete results are in Section 5.

Based on Fugu's results, we believe that continual learning of ABR and congestion-control algorithms *in situ* is a compelling research direction. Accordingly, we plan to operate Puffer for several years and will open it for use by the research community, letting researchers train and test new algorithms on randomized subsets of its traffic.

This paper proceeds as follows. In Section 2, we discuss the background of Internet video streaming and compare Fugu with related work. We then describe Fugu's design and implementation (§3) and the Puffer platform for evaluation and training (§4). In Section 5, we describe the results of the evaluation. We then discuss the limitations of Fugu and of the evaluation (§6) and the ethical and legal implications of our work (§7) before concluding (§8).

2 BACKGROUND AND RELATED WORK

The basic problem of adaptive video streaming has been the subject of much academic work; for a good overview, we refer the reader to Yin et al. [33]. We briefly outline the problem here. A service wishes to serve a pre-recorded or live video stream to a broad array of clients over the Internet. Each client's connection has a different and time-varying capacity for throughput. The initial and future throughput is unknown *a priori*. Because there are many clients, it is not feasible for the service to adjust the encoder configuration in real time to accommodate any one client.

Instead, the service encodes the video into a handful of alternative compressed versions, each representing the original video but at a different quality, target bitrate, or resolution. Most commercial services encode between five and eight alternative versions. Client sessions choose from this limited menu. The service encodes the different versions in a way that allows clients to switch midstream as necessary: it divides the video into *chunks*, typically 2–6 seconds each, and encodes each version of each chunk independently, so it can be decoded without access to any other chunks. This gives clients the opportunity to switch between different alternative versions at each chunk boundary.

Streaming services today generally use "variable bitrate" (VBR) encoding, where within each alternative stream, the chunks vary in compressed size. This yields better quality than constant bitrate (CBR) encoding. Nonetheless, the different alternative versions are often referred to as "bitrates" (referring to their average data rate over time, or to a minimum throughput sufficient to play the stream without underflow, given an initial and a maximum buffer size).

Choosing which chunks to fetch. Algorithms that select which alternative version of each chunk to fetch and play, given uncertain future throughput, are known as *adaptive bitrate* (ABR) schemes. These schemes fetch chunks, accumulating them in a playback buffer, while playing the video at the same time. The playhead advances and drains the buffer at a steady rate, 1 s/s, but chunks arrive at irregular intervals dictated by the varying network throughput and the compressed size of each chunk. If the buffer underflows, playback must stall while the client "rebuffers": fetching more chunks before resuming playback. The goal of an ABR algorithm is typically framed as choosing the optimal sequence of chunks to fetch,[2] given estimates of current throughput and guesses about future throughput, in an effort to minimize startup time, minimize the frequency or duration of stalls, maximize the quality or bitrate of chunks played back, and minimize variation in quality over time (especially abrupt changes in quality). The importance tradeoff for these factors is captured in a QoE metric; several studies have calibrated QoE metrics against human behavior or opinion [3, 9, 13].

How Fugu relates to prior ABR schemes. Researchers have produced a literature of ABR schemes, including "rate-based" approaches that focus on matching the video bitrate to the network throughput [12, 14, 17], "buffer-based" algorithms that steer the duration of the playback buffer [11, 26, 27], and control-theoretic schemes that try to maximize expected QoE over a receding horizon, given the upcoming chunk sizes and a prediction of the future throughput.

Model-Predictive Control (MPC), FastMPC, and RobustMPC [33] fall into the last category: they take as input a *throughput predictor* that informs a predictive *model* of what will happen to the buffer occupancy and QoE in the near future, depending on which chunks it fetches, with what quality and sizes. MPC uses the model to plan a sequence of chunks over a limited horizon (e.g., the next 5–8 chunks) to maximize the expected QoE. We implemented MPC and RobustMPC for Puffer, using the same predictor as the paper: the harmonic mean of the last five throughput samples.

Pensieve [15] uses reinforcement learning (RL) to solve the ABR problem without an explicit model; it teaches a neural network to make ABR decisions by training it in a network simulator over a set of traces, rewarding it for decisions that produced good QoE. This is known as model-free RL.

Fugu fits between MPC and Pensieve in approach. Fugu is based on MPC and uses the same codebase that we developed to implement MPC, and the same model. However, instead of using a throughput predictor, Fugu is based around a *transmission-time predictor* (TTP). The TTP differs from a throughput predictor in four principal ways:

(1) **It accounts for size.** The TTP predicts *how long* it would take the client to fetch a proposed chunk *of a certain size*, rather than throughput. This allows the TTP to account for throughput that varies in time or with chunk size [4].
(2) **It is probabilistic.** The TTP outputs a probability distribution over the time instead of a single point estimate.
(3) **It considers TCP-layer statistics.** The TTP's inputs include the traditional inputs to a throughput predictor (recent chunk sizes and their transfer times), but also sender-side TCP statistics: the congestion window, the number of bytes in-flight, the RTT, and TCP's estimate of the current throughput (delivery rate).
(4) **It is a continually trained neural network.** The TTP is implemented with a neural network, and each day, we retrain the TTP on the past week of Puffer's data.

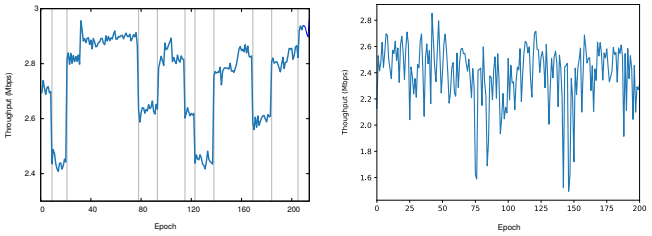
With this design, which combines model-predictive control and reinforcement learning of a neural network, Fugu belongs to a family of approaches known as model-based reinforcement learning [19]. Other than these differences, Fugu is identical to MPC (as evaluated in this study).

Training *in situ*. Pensieve was trained in a trace-based network simulator, and evaluated in emulation and on Internet paths. In this study, we found a substantial difference in performance between the version of Fugu that was trained on Puffer vs. a version trained in emulation, with the same set of traces used by Pensieve [7]. The emulation-trained version of Fugu performed poorly on Puffer. This suggests that either learning *in situ*, or improving the fidelity of Internet emulators, may be necessary to obtain good real-world performance from learned networking algorithms.

Continual learning for networking. We follow prior work in networking that uses continual learning in deployment. TCP WISE [21] divides users into clusters and experiments with different initial windows for each cluster to learn the value that minimizes the latency of an initial HTTP exchange. It was evaluated on a front-end server at the Baidu search engine, learning week-by-week, and reduced initial HTTP latency by 8% overall. CS2P [29] also divides users into clusters, and learns an ABR throughput predictor for each cluster based on the observation that throughput tends to jump over the course of a connection between a finite number of states. In addition to emulation evaluations, CS2P was used over the Internet in a four-day test, retraining daily, with 200 volunteer clients at universities; it improved QoE by 3.2% (improving average bitrate and stalls, while slightly worsening variability). We have not observed CS2P's discrete throughput states in our dataset (Figure 2).

Fugu and Puffer build on this work, extending its scope and magnitude. The TTP was trained daily from the prior week of experience with an uncontrolled population of 3,700

[2] Some systems can also replace a fetched chunk with a better version [26].



(a) CS2P figure 4(a): an example session

(b) Typical Puffer session with similar throughput

Figure 2: Puffer has not observed CS2P's discrete throughput states. (Epochs are 6 s in both plots.)

users. We demonstrate a substantial benefit from continual learning, compared with a version of Fugu that was trained just once on Puffer's data from January 1–15, 2019.

3 FUGU: DESIGN AND IMPLEMENTATION

Fugu is a model-based reinforcement learning algorithm for bitrate selection that trains with continual learning on real user data. As it streams a video to a client, Fugu selects which sized chunks (bitrate) to send in order to optimize the QoE over the next N seconds. More formally, it models adaptive video streaming as a stochastic optimal control problem, with the goal of maximizing the cumulative QoE over a finite horizon.

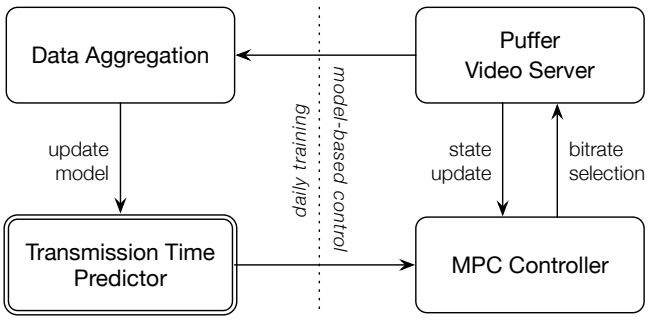
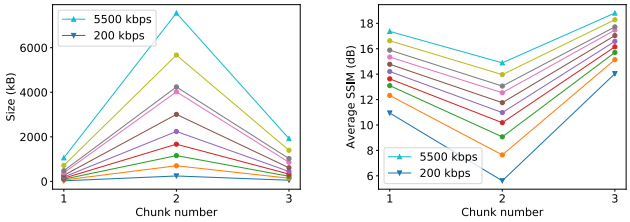


Figure 3: Overview of Fugu

Figure 3 shows Fugu's high-level design. So that it can easily update its model and aggregate performance data from many streams over long time periods, Fugu runs on the server. In cases when client information is needed, clients send that information to the server.

A model predictive controller decides which size of each chunk to send to each client. It sends these decisions to a media server, which is responsible for sending the chunk and reporting data on network behavior back to the controller.

The controller, described in Section 3.4, is a reinforcement learning system that uses current network state, a novel



(a) VBR encoding lets chunk size vary, even within a bitrate.

(b) Picture quality also varies within each bitrate.

Figure 4: Variations in picture quality within each bitrate demonstrate the advantages of choosing chunks based on SSIM and size, rather than "bitrate."

Transmission Time Predictor (TTP), and the objective function (Section 3.1) to decide which chunks to send. The TTP, described in Section 3.2, is a neural network trained on real user data to estimate the transmission time for a given chunk.

3.1 Objective function

For each video chunk K_i, Fugu has a selection of different sizes S to choose from, K_i^s, where $s \in S$. As with prior approaches, Fugu quantifies the QoE of each chunk as a linear combination of video quality, video quality variation, and stall time [33]. Unlike some prior approaches, which use the compressed bitrate as a proxy for image quality, Fugu optimizes a perceptual measure of picture quality—in our case, SSIM. This has been shown to correlate with human opinions of QoE [9]. We emphasize that we use the same objective function in our version of MPC and RobustMPC as well—all three schemes optimize for SSIM in the same way. Figure 4 demonstrates the advantages of this approach on three adjacent sets of real-world chunks from Puffer.

Let $Q(K)$ be the video quality of a chunk K, $T(K)$ be the uncertain transmission time of K, and B_i be the current playback buffer size. Fugu defines the QoE of K_i^s as

$$QoE(K_i^s) = Q(K_i^s) - \lambda|Q(K_i^s) - Q(K_{i-1})| - \mu(T(K_i^s) - B_i)_+, \quad (1)$$

where λ and μ are configuration constants for how much to weight video quality variation and rebuffering. The $+$ operator is defined as $(x)_+ = \max\{x, 0\}$, such that it captures only the positive part of x. The term $(T(K_i^s) - B_i)_+$ therefore describes the stall time experienced by sending K_i^s. This is the same formulation used in MPC [33]. Fugu plans a trajectory of sizes s of the future H chunks to maximize their total QoE.

3.2 Transmission-Time Predictor (TTP)

Once Fugu decides which size of a chunk to send, the video quality and video quality variation are known. The only uncertainty is the stall time. Since the server knows the current playback buffer size, all it needs to compute the stall

time is the transmission time: how long will it take for the client to receive the chunk? If the server has an oracle that reports the transmission time of each chunk, it can compute the optimal plan that maximizes QoE.

Fugu uses neural-network transmission-time predictors to approximate the oracle. For each chunk in the fixed horizon it is optimizing for, it trains a separate predictor. E.g., if optimizing for the total QoE of the next five chunks, it trains five neural networks. Multiple networks in parallel are functionally equivalent to one network that takes which future time step as a variable, and we have observed equivalent performance. Fugu trains multiple models so that it can parallelize training without data parallelism and can increase the horizon length without retraining previous models.

Each TTP network takes its input as a 4-vector of:

(1) sizes of past t chunks: $K_{i-t}, \ldots, K_{i-1}$,
(2) transmission times of past t chunks $T_{i-t}, \ldots, T_{i-1}$;
(3) internal TCP statistics (Linux `tcp_info` structure)
(4) size of the chunk to be transmitted.

The TCP statistics include the current congestion window size, the number of unacknowledged packets in flight, the current RTT, the minimum RTT, and the estimated throughput (`tcpi_delivery_rate`).

Prior approaches have used Harmonic Mean (HM) [33] or a Hidden Markov Model (HMM) [29] to predict a single throughput for the entire lookahead horizon. In contrast, the transmission-time predictor outputs a probability distribution $\hat{T}(K_i^s)$ over the transmission time of K_i^s.

It explicitly considers the chunk size of K_i into the model because that is fundamentally more powerful than a single throughput estimate. For example, consider an example network trace as (Figure 5) in which the controller can send either a 1Mb or a 2Mb chunk for 2 seconds of video. An instantaneous throughput estimate at time 0 will return 1Mbps, suggesting that the controller should select the 2Mb chunk. However, if it does, after 1 second the throughput goes to zero and the chunk is never completed. A TTP could estimate the transmission time of the 1Mb chunk as 1s and the 2Mb chunk as infinite. In this case, no throughput estimate can estimate the transmission times of both chunks accurately.

Similarly, TTP outputs a probability distribution given a chunk because it is fundamentally more powerful than a single transmission-time estimate as well. If a bandwidth enforcer or an emulation trace varies the throughput with a random walk, no single estimate will be able to capture the stochastic process. We quantify the advantage of TTP by comparing it with HM on Puffer in the real world (§5).

3.3 Training the TTP

The TTP collects training data $\mathcal{D}$ by running ABR algorithms on real users, aggregating pairs of (a) the input 4-vector and, (b) the true transmission time for the chunk. It discretizes transmission time to simplify the regression problem the neural network to learn.

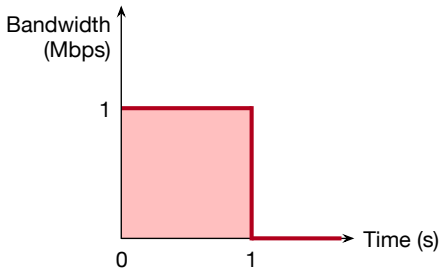


Figure 5: An intuitive example showing the benefits of considering the chunk size in prediction: if the throughput prediction is as shown, chunks larger than 1 megabit will never finish downloading.

TTP trains on $\mathcal{D}$ with standard supervised learning: the training minimizes the cross-entropy loss between the output probability distribution and the discretized actual transmission time using stochastic gradient descent.

A typical challenge is that a mismatch between the training distribution $\mathcal{D}$ and the distribution observed by Fugu's controller will lead to performance degradation [25]. In addition, the real environment may drift over time; learning with the most recent data does not necessarily work because of catastrophic forgetting. Fugu uses both reinforcement learning and continual learning to address these issues.

3.4 Model-based controller

Fugu's model-based controller is a stochastic optimal controller that maximizes the cumulative QoE in Equation 1. It queries TTP for predictions of transmission time and outputs a plan $K_i^s, K_{i+1}^s, \ldots, K_{i+H-1}^s$ by value iteration [5]. After sending K_i^s, the controller observes and updates the input vector passed into TTP, and replans again for the next chunk.

Given the current playback buffer level, let $v_i^*(B_i)$ denote the maximum QoE that can be achieved in the H-step lookahead horizon. We have value iteration as follows:

$$v_i^*(B_i) = \max_{K_i^s} \left\{ \sum_{T_i} \Pr[\hat{T}(K_i^s) = T_i](QoE(K_i^s) + v_{i+1}^*(B_{i+1})) \right\},$$

where $\Pr[\hat{T}(K_i^s) = T_i]$ is the probability for TTP to output a discretized transmission time T_i, and B_{i+1} can be derived by system dynamics once T_i is estimated. The controller computes the optimal trajectory by solving the above value iteration with dynamic programming (DP). To make the DP computational feasible, it discretizes B_i into bins and uses forward recursion with memoization to speed up the computation by avoiding recomputation of $v_i^*(B_i)$.

Broadly speaking, Fugu's model-based controller falls into the class of model-predictive control because of its replanning at every step. The prior work on MPC for video streaming [33] employed a different style of model-predictive control ("certainty-equivalent MPC"), informed by the point estimate supplied by the throughput predictor. By contrast, Fugu takes the TTP's "fuzzy" prediction into account.

3.5 RL and continual learning

We improve Fugu with reinforcement learning, by repeatedly: 1) running Fugu with the current TTP on real users to collect training data; 2) appending the new data to $\mathcal{D}$; 3) retraining the TTP with $\mathcal{D}$. RL helps mitigate the distributional mismatch between $\mathcal{D}$ and the model-based controller [25].

However, training on a constantly growing $\mathcal{D}$ is not feasible in practice; meanwhile, using a bad sampling strategy may lead to "catastrophic forgetting" [16], the problem that neural networks trained with new data may lose the parameters to perform well on the old data. Therefore, we apply a technique in continual learning known as "experience replay" [23, 24] to mitigate catastrophic forgetting.

Our experience-replay strategy is relatively simple: we run Fugu along with other ABR algorithms on real users for a whole day to collect both on-policy and off-policy data, in which the off-policy data collected from other ABR algorithms help Fugu observe more diverse and real input. Next, Fugu samples a fixed amount of training data from the past week that is sufficient and computationally feasible to train on. In particular, Fugu assigns a different weight of $\gamma^n, \gamma \in (0, 1]$ to the n-th day in the past, to prioritize more recent data and replay old data. Fugu also shuffles the sampled data to remove correlation in the sequence of inputs. The weights from the previous model are loaded to warm-start the retraining of the neural network.

3.6 Implementation

TTP takes as input the past $t = 8$ chunks, and outputs a probability distribution over 21 bins of transmission time: $[0, 0.25), [0.25, 0.75), [0.75, 1.25), \ldots, [9.75, \infty)$, with 0.5 seconds as the bin size except for the first and the last bins. TTP is a fully-connected neural network, with two hidden layers with 64 neurons each. We tested different TTPs with various numbers of hidden layers and neurons, and found similar training losses across a range of conditions for each. We implemented TTP and the training in PyTorch, but we load the trained model in C++ when running on the production server for performance. A forward pass of TTP's neural network in C++ imposes minimal overhead per chunk (less than 0.3 ms on average on a recent x86-64 core).

Fugu's model-based controller optimizes over $H = 5$ future steps (about 10 seconds), and performs forward recursion with memoization after discretizing the buffer size into the same bins as TTP. We set $\lambda = 1$ and $\mu = 100$ to balance the conflicting goals in QoE. During continual learning, we retrain TTP automatically day by day on the sampled data from the past 7 days. We set $\gamma = 0.8$ and sample 1 million pairs of input and output data at most (i.e., with 25.3% of data from the previous day and 6.6% of data from a week ago). Each retraining takes about 3 hours to complete.

4 PUFFER: A PUBLIC ABR TESTBED

To obtain sufficient data to train Fugu daily in a real-world environment, and to evaluate Fugu and other ABR schemes, we built Puffer, a free, publicly accessible website that live-streams six over-the-air broadcast television channels. Here, we describe the design of the back-end system, communication with the browser, implementation and hosting of the ABR algorithms, and some statistics about the userbase.

4.1 Back-end: decoding, encoding, SSIM

Puffer receives six television channels using a VHF/UHF antenna and an ATSC demodulator, which outputs MPEG-2 transport streams in UDP. We wrote software to decode a stream to chunks of raw decoded video and audio, maintaining synchronization (by inserting black fields or silence) in the event of lost transport-stream packets on either substream. Video chunks are 2.002 seconds long, including the 1/1000 factor for NTSC frame rates. Audio chunks are 4.8 seconds long. Video is de-interlaced with `ffmpeg` to produce a "canonical" 1080p60 or 720p60 source for compression.

Puffer encodes each canonical video chunk in ten different H.264 versions, using `libx264`. The encodings range from 240p60 video with constant rate factor (CRF) of 26 (about 200 kbps) to 1080p60 video with CRF of 20 (about 5,500 kbps). Audio chunks are encoded in the Opus format.

Puffer then uses `ffmpeg` to calculate the SSIM [31] of each encoded chunk relative to the canonical source. This information is used by the objective function (QoE metric) of MPC, RobustMPC, and Fugu, and for our evaluation. After all of the versions of a chunk have been encoded and each SSIM is known, Puffer allows the chunk to be served. Chunks are held in shared memory and available for about 30 seconds before they are deleted; clients may buffer up to 15 seconds.

4.2 Serving chunks to the browser

To make it feasible to deploy and test arbitrary ABR schemes, Puffer uses a "dumb" player (using the HTML5 `<video>` tag and the JavaScript MediaSource extensions) on the client side, and places the ABR scheme at the server. We have a 48-core server with 10 Gbps Ethernet in a well-connected datacenter. The browser opens a WebSocket (TCP) connection to the server, where it is accepted by one of 48 serving daemons

(one per core). Each daemon is configured with a different combination of TCP congestion control and ABR scheme. The daemons all listen on the same TCP port using Linux's SO_REUSEPORT feature, and depend on the kernel to spread out incoming connections across the daemons, and therefore across ABR/congestion-control combinations.[3]

A broadly similar approach was used in Pensieve [15, 33]; the ABR scheme was implemented in an off-client agent that told the client what chunks to fetch. Puffer takes this server-side architecture even further. Over the WebSocket, the browser tells the server when it receives each video chunk and periodically updates the server about the duration of its playback buffer. The server chooses which chunks to send the client, and sends them over the WebSocket over TCP.

This design for experimentation means that Puffer is not a DASH [18] (Dynamic Adaptive Streaming over HTTP) system—the predominant implementation of ABR streaming video today. In Puffer, the ABR algorithm does not run in the client, and the client does not make an individual HTTP request for each chunk. Like DASH, however, Puffer is an ABR system streaming chunked video over a TCP connection, and it runs the same ABR algorithms that DASH systems can run. We don't expect this architecture to replace client-side ABR (which can be served by CDN edge nodes), but we expect its conclusions to translate to ABR schemes broadly.

4.3 Hosting arbitrary ABR schemes

We implemented buffer-based control, MPC, RobustMPC, and Fugu in back-end daemons that serve video chunks over the WebSocket. Compared with the original MPC/RobustMPC, we replaced the use of "bitrate" in the objective function with SSIM (we also used the same objective function for Fugu). For buffer-based control, we used the formula in the original paper [11] to choose reservoir values consistent with our 15-second maximum buffer duration.

Deploying Pensieve for live streaming. Reimplementing Pensieve, however, would have been more difficult. Given how difficult it can be to get machine-learning approaches exactly right, we opted to use the released Pensieve code (written in Python with TensorFlow) directly. When a client is assigned to Pensieve, Puffer spawns a Python subprocess running Pensieve's multi-video model. Communication with this subprocess occurs over a Unix-domain socket.

We contacted the Pensieve authors to request advice on deploying the algorithm in a live, multi-video, real-world setting. The authors recommended that we use a longer-running training and that we tune the entropy parameter when training the multi-video neural network. We wrote an automated tool to train several different models. We tested these manually, then selected the model with the best performance. We modified the Pensieve code to set `video_num_chunks` to a large value, so that Pensieve never thinks the video will end. We changed the "bitrate ladder" to match the average bitrates of the channels in our system. We adjusted the video chunk length to 2 seconds and the buffer threshold to 15 seconds to reflect the Puffer parameters. For training data, we used the same simulated videos used to create the multi-video model presented in their paper, and a combination of the FCC and Norway traces provided with Pensieve's code.

4.4 Puffer's userbase

The Puffer website works in the Chrome, Firefox, and Edge browsers, including on Android phones. It does not play in the Safari browser (which doesn't support Opus audio in MediaSource) or on Mobile Safari (which doesn't support MediaSource or DASH at all).

To recruit participants, we purchased Google and Reddit ads for keywords such as "live tv" and "tv streaming," and paid people on Amazon Mechanical Turk to stream video from Puffer. Approximately 25% of the users arrived via news articles and blog posts, 25% from Google ads, 20% from Reddit ads, 20% from a search engine, and 10% from Mechanical Turk. From opening in December 2018 to January 28, 2019, Puffer has streamed 12,080 hours of video to 6,402 distinct user accounts. We deployed the continual-learning version of Fugu on January 19, and began the randomized study. In the nine days between January 19 and January 28, Puffer streamed 8,131 hours of video to 3,719 user accounts, split between Cubic [10] and BBR [6] congestion control. Every ABR scheme was used for at least 1,170 hours of streaming.

5 EVALUATION

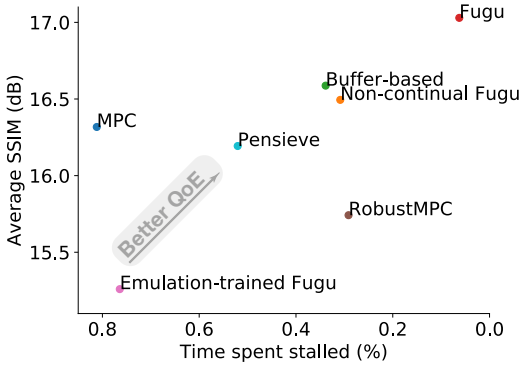
We now present our experimental evaluation of Fugu on Puffer. The results support the following findings:

(1) Fugu outperforms prior ABR algorithms in delivering high QoE to end users, as evaluated by stall time, mean SSIM, and SSIM variability.
(2) Congestion-control information is helpful for generating accurate predictions of transmission times.
(3) In-situ training of ABR algorithms improves performance.
(4) Continual learning improves performance.

[3] We did not collect equal amounts of data, i.e., hours streamed, across each scheme, for several reasons. We suspect Linux does not perfectly randomize incoming TCP connections to SO_REUSEPORT listening sockets. In addition, because TCP connections are assigned to listening sockets once on arrival, schemes that caused users to stream for longer per session accumulated more data. In addition, each daemon stops listening when it has 10 streams, meaning Puffer refuses to accept connections at all after it reaches 480 simultaneous streams (motivated by the limits of 10 Gbps Ethernet and to avoid congestion at the server). Overall, the total amount of hours streamed during the nine-day analysis period is within 30% between schemes (all ABR schemes had between 1,170 and 1,486 hours of data collected).

Over BBR congestion control

Algorithm	Time stalled (lower is better)	Mean SSIM (higher is better)	SSIM variation (lower is better)
Fugu	0.06%	17.03 dB	0.53 dB
Buffer-based	0.34%	16.58 dB	0.79 dB
MPC	0.81%	16.32 dB	0.54 dB
RobustMPC	0.29%	15.74 dB	0.63 dB
Pensieve	0.52%	16.19 dB	0.88 dB
Non-continual Fugu	0.31%	16.49 dB	0.60 dB
Emulation-trained Fugu	0.76%	15.26 dB	1.03 dB



Over Cubic congestion control

Algorithm	Time stalled (lower is better)	Mean SSIM (higher is better)	SSIM variation (lower is better)
Fugu	0.08%	16.88 dB	0.58 dB
Buffer-based	0.28%	16.72 dB	0.77 dB
MPC	0.53%	16.30 dB	0.53 dB
RobustMPC	0.38%	15.50 dB	0.62 dB
Pensieve	0.76%	15.64 dB	1.04 dB
Non-continual Fugu	0.80%	15.54 dB	0.70 dB

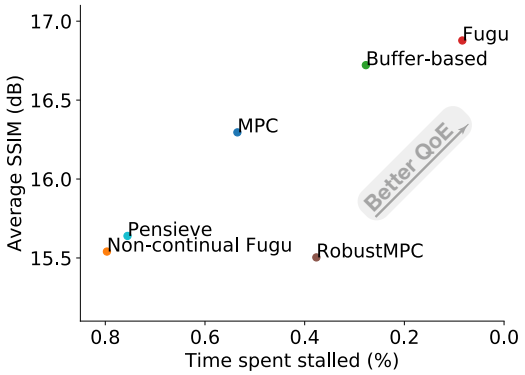
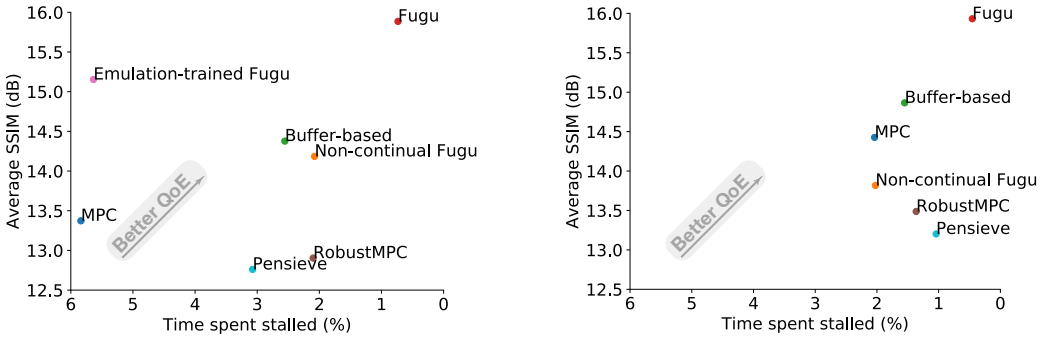


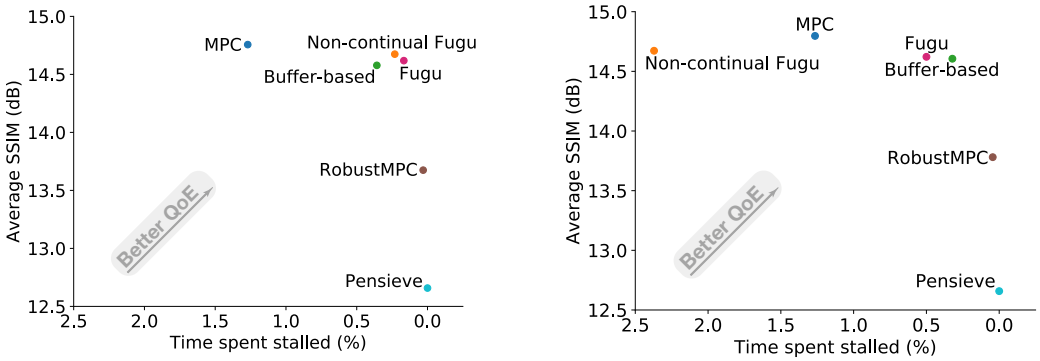
Figure 6: In a real-world randomized evaluation from January 19–28, Fugu substantially outperformed the other ABR algorithms. It reduced the time spent stalled by 4–13×, increased SSIM, and reduced chunk-to-chunk variation in SSIM. Fugu outperformed (by at least 5× on stalls) a "non-continual" version of itself that was trained on Puffer's data collected from January 1–15, but never retrained subsequently—demonstrating the value of continual learning. It also outperformed a version of itself trained "ex situ" in emulation on the FCC dataset [7].



(a) BBR **(b)** Cubic

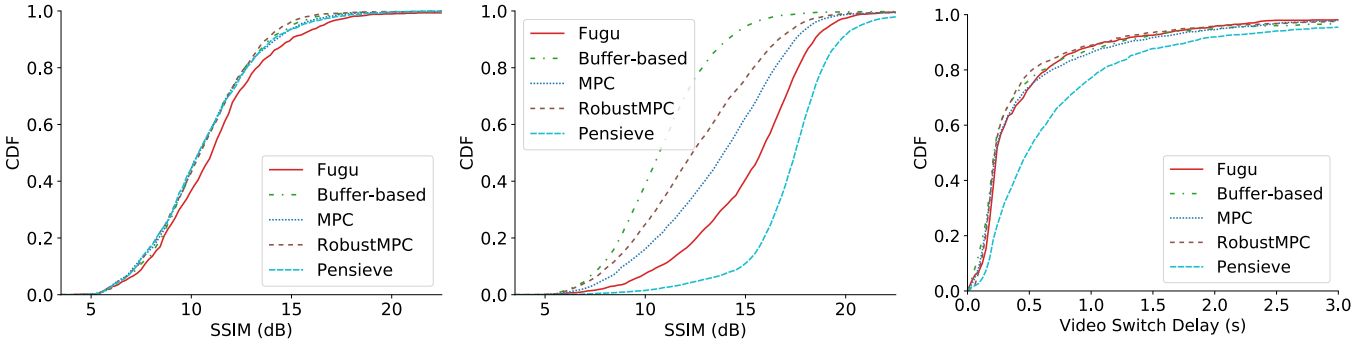
Figure 7: Focusing on results from sessions with mean throughput of less than 6 Mbps (following prior work [15, 33], these are more likely to demand nontrivial answers from an ABR algorithm), Fugu also had superior performance.

(5) Schemes with fewer stalls and better SSIM were associated with users choosing to stream for a longer period of time. Users assigned to Fugu waited twice as long, on



(a) BBR **(b)** Cubic

Figure 8: Results in emulation (run in mahimahi [20] using the FCC traces [7], following the method of Pensieve [15]). In this situation, emulation results do not do a good job predicting real-world performance, either of the "all-comers" dataset (Figure 6) or the users connected over slower network paths (Figure 7).



(a) First chunk SSIM (cold start) **(b)** First chunk SSIM (channel change) **(c)** Startup delay (channel change)

Figure 9: Transient behavior on startup and channel change. Schemes are essentially at parity on a cold start, but on a channel change (keeping the same TCP connection and ABR state, but changing the video), Fugu and Pensieve deliver superior initial picture quality (SSIM)—Pensieve even better than Fugu, but at the cost of increased delay.

average, before quitting or reloading the page, compared with MPC, RobustMPC, and Pensieve.

5.1 Real-world performance summary

Puffer ran a parallel randomized study of 13 combinations of ABR and congestion control algorithms over a nine-day period in January 2019, streaming 8,131 hours of video to 3,719 unique users over the Internet. Our collection data includes all activity Puffer received over this period, excluding sessions that streamed less than 5 seconds of video, and counting sessions stalled for longer than 30 seconds as dead. Figure 6 shows the overall results: Fugu substantially outperforms the previous algorithms in almost all metrics collected. The cleanest comparison is with MPC, because these share the same optimization goal and control strategy. The use of a continual learning TTP instead of the HM throughput predictor reduced stalls by 13× (BBR) or 7× (Cubic), while also improving SSIM and maintaining SSIM variability.

The results also demonstrate clearly the effect of *continual* learning. Fugu substantially outperforms a version of itself that is identical except for its training regimen: "non-continual" Fugu was trained once on data from January 1–15. Unlike Fugu, it was not then retrained daily. Stalls are 5–13× more frequent. The accuracy of the "non-continual" TTP, in the face of a dynamic and growing user population joining from new locales and networks, was not adequate.

5.2 QoE metrics predict stream duration

As shown in Figure 10, we observed statistically significant differences in the session durations of users across algorithms. When compared with MPC, RobustMPC, and Pensieve, streams randomly assigned to Fugu lasted more than twice as long, on average, before the user quit or reloaded,

Algorithm (any congestion control)	Stream duration (mean ± std. err.) (higher may suggest more-satisfied users)
Fugu	20.3 ± 1.3 minutes
Buffer-based	12.9 ± 0.8 minutes
MPC	9.5 ± 0.5 minutes
RobustMPC	10.0 ± 0.5 minutes
Pensieve	10.0 ± 0.5 minutes
Non-continual Fugu	12.5 ± 0.7 minutes
Emulation-trained Fugu	7.9 ± 1.3 minutes

Figure 10: Mean session durations before the user quit or reloaded the page. Schemes with fewer stalls and better SSIM are associated with longer mean duration.

TTP ablation	Prediction error rate (lower is better)
No History + No cwnd inflight	19.8%
No History + No delivery rate	17.9%
No History + No RTT	24.0%
No History	17.1%
Full TTP	13.7%
Harmonic Mean	17.9%

Figure 11: TTP ablation study (BBR). Several components of the `tcp_info` structure prove helpful in predicting the transmission time of proposed chunks in the future.

and about 58% longer than with buffer-based control. The results suggest—but hardly establish—that users who received a blinded assignment to Fugu were more likely to be satisfied or less frustrated with the quality of the stream.

Differences in session duration were correlated with decreased stall time and increased SSIM. In a weighted multivariate linear regression, each extra percent of time stalled was associated with a decrease in the average stream duration by about 6 minutes, and each extra decibel of SSIM predicted an increase in average stream duration by about 2 minutes. These values are subject to substantial statistical and systematic uncertainty, but may be useful in establishing an overall QoE metric as a function of the metrics measured here. The R^2 goodness-of-fit metric was 0.36, suggesting the multivariate linear model is predictive but hardly definitive. Others have more carefully studied this issue in slightly different formulations [3, 9, 13].

5.3 Emulation vs. real-world results

Each of the ABR algorithms we evaluate alongside Pensieve was evaluated in emulation in prior works [15, 33]. Notably, the results in those works are qualitatively different than some of the real world results we have seen here—for example, buffer-based control outperforms MPC, RobustMPC, and Pensieve, contrary to two prior emulation studies.

To investigate this further, we constructed an emulation environment similar to that used in [15]. This involved running the Puffer media server locally, and launching headless Chrome clients inside mahimahi [20] shells to connect to the server. Each mahimahi shell imposed a 40 ms end-to-end delay on traffic originating inside it and limited the downlink capacity over time to match the capacity recorded in a set of FCC broadband network traces [7]. As in the Pensieve evaluation, uplink speeds in all shells were capped at 12 Mbps. Within this test setup, we automated 12 clients to repeatedly connect to the media server, which would play a 10 minute clip recorded on NBC over each network trace in the dataset. Each client was assigned to a different combination of ABR and CC algorithms, and played the 10 minute video repeatedly over more than 15 hours of FCC traces. The results from this experiment are depicted in Figure 8.

Comparing Figure 8 to Figure 6 is illuminating—the emulation results do not look even close to the real world results. In emulation, almost every algorithm tested lies somewhere along the SSIM/stall frontier, with Pensieve rebuffering the least and MPC delivering the highest quality video, and the other algorithms lying somewhere in between. In the real world, we see a clear performance order, with Fugu clearly outperforming other algorithms in quality and stall time, and with Buffer-based coming in second in both metrics. These results suggest research opportunities in constructing network emulators that capture additional dynamics of the real Internet that are not currently being captured.

We also note that the version of Fugu that was *trained* in this emulated environment, then evaluated in the real world, also performed poorly—again, a demonstration of the weakness of current emulators (or, at least, the FCC dataset) to predict performance over Internet paths of the kind used in Puffer.

5.4 Contributors to TTP accuracy

As evaluated here, Fugu is identical to MPC, but for the use of the TTP instead of a throughput predictor. To quantify the primary source of these gains, we perform an ablation study. In this study, we calculate what a set of ablated transmission time predictors would have predicted in response to a new sequence of input data encountered by Puffer. The ablated predictors were trained on prior data which did not include the testing data used to evaluate it. For each predictor, we evaluate the percentage of time that the actual transmission time recorded falls within the highest probability bin provided by the TTP. The ablated predictors we used in this experiment were a TTP without congestion window knowledge, a TTP without knowledge of the TCP-provided "delivery rate", a TTP without RTT knowledge, and a TTP

with knowledge of only one chunk in the past instead of 8. We also calculated the accuracy that would have been achieved by a harmonic mean throughput estimator such as the one used in MPC. Figure 11 contains the result of this study.

Analyzing these results, it can be seen that each of the inputs to the TTP—each of the TCP statistics in the `tcp_info` structure, and the recent history—contributes positively to its accuracy. The results suggest that IP-layer RTT measurements (of the kind taken by TCP to produce an SRTT estimate) are the most helpful single signal.

5.5 Remarks on Pensieve's performance

One of the more surprising results of our evaluation is the performance of Pensieve relative to MPC, RobustMPC, and buffer-based control. In the original Pensieve paper [15], the authors demonstrated that Pensieve outperformed all three of these algorithms in both simulation-based tests and in video streaming tests on real world networks. In our system, however, we observed that Pensieve+BBR was outperformed by Buffer-based+BBR, and that Pensieve+Cubic performed worse than RobustMPC+Cubic, MPC+Cubic, and Buffer-based+Cubic. We believe this mismatch in results occurred for several reasons.

First, we have found that emulation-based training and testing (or, at least, mahimahi tests with the FCC dataset) do not capture the vagaries of the real-world paths seen in the Puffer study.

Second, most of the evaluation of Pensieve in the original paper focused on evaluating Pensieve using a single test video. As a result, that model may have been able to learn characteristics of the video itself, and apply these when making decisions. The authors did also train a model on multiple videos, and this is the model which we use in our testing environment. While the multi-video model in the original paper performed only 3.2% worse than their single-video model, the original paper only evaluated this against the original video. The videos that we play vary significantly over time, with programming ranging from slow-moving talk shows to fast-moving sports content, a level of diversity which the simulated videos used to train the Pensieve multi-video model may struggle to capture.

Third, the initial Pensieve paper focused on the problem of streaming on-demand video, while our evaluation focused on live video. As described in the previous section, this mismatch meant that we had to modify the training and evaluation of Pensieve to fit our use case, which involved removing one variable that Pensieve used when making decisions—the length of the video being played. As a result, in our tests, Pensieve could not realize any gains over other algorithms by planning for the end of playback.

Fourth, while the Pensieve authors did make their code publicly available, they note that they only provide a subset of the data which they used to train the model that they present in their paper. As a result, it is possible that the model that we trained on the data they provided was less well-informed than the model that they tested in their paper.

Finally, the Pensieve authors based their QoE evaluation around bitrate as an evaluation metric. As we have shown in Figure 4, a bitrate-based optimization may not map well to a picture-quality-based evaluation such as ours. As a result, Pensieve may have suffered from its internal reward function optimizing bitrate while our external evaluation was based on SSIM.

We would like to emphasize that our findings do not indicate that Pensieve cannot be a useful ABR algorithm—in a scenario where similar, pre-recorded video is being played over a familiar set of networks, Pensieve represents a high-performing algorithm capable of delivering high QoE without the costs of continual learning. Pensieve's performance in our evaluation may suggest that offline learning on emulation is insufficient for systems which must stream live, highly variable video content to an uncontrolled and growing user population on diverse, dynamic networks.

6 LIMITATIONS AND FUTURE WORK

The design of Fugu, and the evaluation we conducted here on Puffer, are subject to important limitations that may affect their performance and generalizability.

6.1 Limitations of the evaluation

The gains seen on Puffer are *statistically* significant and represent a substantial performance improvement, but may have a substantial *systematic* bias that is hard to control for. We only deployed the continual learning version of Fugu on January 19, not long before the submission deadline for this manuscript (which reports an analysis conducted up through the last feasible day, January 28). These gains occurred while Puffer's user population was rapidly expanding with new users from the public. It is hard to predict whether the gains seen here will persist over the coming months or years, as the user population stabilizes and consists mostly of returning users instead of new users. For example, perhaps continual learning will not be as necessary when it is possible to statically train once on a year of data. We have continued to operate Puffer and the randomized experiment and will be able to report a much longer experiment in subsequent versions of this paper.

Puffer's server-side design was built for the needs of an academic experiment wishing to centrally experiment with and randomize users to different congestion-control and

ABR schemes (many of which are CPU-intensive and implemented in C++, and with visibility into the server-side congestion-control statistics). It is unknown to what degree these results translate into the more common client-side DASH-to-CDN setting, used by services like YouTube, Netflix, and Twitch. We are optimistic that the basic principles of chunk-by-chunk ABR-over-TCP are shared across these contexts, but have not demonstrated or evaluated this.

It is also unknown to what degree Puffer's results—which are about a *single* server with 10 Gbps connectivity in a well-provisioned datacenter, sending to clients across our entire country over the wide-area Internet—generalize to the more typical paths between a user on an access network and their nearest CDN edge node.

We relied on the Linux kernel to randomize incoming connections to different daemons (each representing a different ABR/congestion-control combination). We do not think this randomization is perfect, and it is theoretically possible that it has biased the analysis—e.g., if Linux's assignment of connection to socket were somehow correlated with other performance factors, such as the client's RTT or IP prefix. We intend to improve (or at least study) the quality of randomization in future work.

Some of Fugu's performance (and that of MPC and RobustMPC) relative to buffer-based control and Pensieve may be due to the fact that these three schemes received more information as they ran—namely, the SSIM of each possible version of each future chunk—than did Pensieve and buffer-based control. It is possible that an "SSIM-aware" Pensieve might perform better. We tried to approximate this sort of scheme (an SSIM-aware neural network trained in emulation or statically) with the "Non-continual Fugu" and "Emulation-trained Fugu" baseline comparators.

6.2 Limitations of Fugu

There is a sense that data-driven algorithms that more "heavily" squeeze out performance gains may also put themselves at risk to brittleness when a deployment environment drifts from one where the algorithm was trained. In that sense, it is hard to say whether Fugu's performance might decay catastrophically some day. We have not found a quantitative justification for *daily* retraining, or a reason to be sure that some surprising detail tomorrow—e.g., a new user from an unfamiliar network—won't send Fugu into a tailspin before it can be retrained. In that sense, even with continual learning, we can't rule out the possibility that Fugu may be more brittle than a simpler algorithm such as buffer-based control. (A nine-day, 3,700-person study is not sufficient to make claims about long-term stability in the face of a changing userbase.)

Fugu does not consider several issues that other research has concerned itself with—e.g., being able to "replace" already-downloaded chunks in the buffer with higher quality versions [26], or optimizing the joint QoE of multiple clients who share a congestion bottleneck.

Fugu is not tied as tightly to the TCP or congestion control as it might be—for example, Fugu could wait to send a chunk until the TCP sender tells it that there is a sufficient congestion window for most of the chunk (or the whole chunk) to be sent immediately. Otherwise, it *might* choose to wait and make a better-informed decision later. Fugu does not schedule the transmission of chunks—it will always send the next chunk as long as the client has room in its playback buffer.

7 ETHICAL AND LEGAL ISSUES

The Institutional Review Board at our institution reviewed a proposal for Puffer and determined that it did not meet the definition of human subjects research and did not require further review. Puffer was also reviewed and approved by our institution's copyright attorneys. Our study benefits, in part, from a law that allows nonprofit organizations to retransmit over-the-air television signals without charge. Another streaming website that makes use of the same provision has published a discussion of the issue [1].

8 CONCLUSION

We have described Fugu, a continual learning algorithm for bitrate selection in streaming video. Fugu learns the best way to send streaming video to diverse real-world clients, mitigating the risk of model mismatch and dataset drift by retraining continually and in the same environment where it is deployed. We evaluated Fugu in a nine-day randomized, blinded study on Puffer, a public website we built that streams live TV to members of the public. Over 8,131 hours of video streamed to 3,719 users, Fugu reduced stalls by 5–13× when compared with other algorithms, while also improving picture quality and reducing quality variation. Fugu and Puffer are open-source software; the repository of code and data is at https://github.com/StanfordSNR/puffer.

We plan to operate Puffer for several years and will open it for use by the research community, letting researchers train and test new congestion-control and ABR algorithms on its traffic. We believe that Puffer could serve as a helpful "medium-scale" stepping-stone for new algorithms, partway between the flexibility of network emulation and the vastness of data—but also conservatism about deploying new algorithms—of commercial services. We do not know what algorithms our colleagues might design given daily feedback about their performance, but we are eager to see and learn.

REFERENCES

[1] 2018. Locast: Non-Profit Retransmission of Broadcast Television. (June 2018). https://www.locast.org/app/uploads/2018/11/Locast-White-Paper.pdf.

[2] Zahaib Akhtar, Yun Seong Nam, Ramesh Govindan, Sanjay Rao, Jessica Chen, Ethan Katz-Bassett, Bruno Ribeiro, Jibin Zhan, and Hui Zhang. 2018. Oboe: auto-tuning video ABR algorithms to network conditions. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. ACM, 44–58.

[3] Athula Balachandran, Vyas Sekar, Aditya Akella, Srinivasan Seshan, Ion Stoica, and Hui Zhang. 2013. Developing a Predictive Model of Quality of Experience for Internet Video. *SIGCOMM Comput. Commun. Rev.* 43, 4 (Aug. 2013), 339–350. https://doi.org/10.1145/2534169.2486025

[4] Mihovil Bartulovic, Junchen Jiang, Sivaraman Balakrishnan, Vyas Sekar, and Bruno Sinopoli. 2017. Biases in Data-Driven Networking, and What to Do About Them. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks (HotNets-XVI)*. ACM, New York, NY, USA, 192–198. https://doi.org/10.1145/3152434.3152448

[5] Richard Bellman. 1957. A Markovian decision process. *Journal of Mathematics and Mechanics* (1957), 679–684.

[6] Neal Cardwell, Yuchung Cheng, C Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. 2016. BBR: Congestion-based congestion control. *Queue* 14, 5 (2016), 50.

[7] Federal Communications Commission. [n. d.]. Measuring Broadband America. ([n. d.]). https://www.fcc.gov/general/measuring-broadband-america.

[8] Paul Crews and Hudson Ayers. 2018. CS 244 '18: Recreating and Extending Pensieve. (2018). https://reproducingnetworkresearch.wordpress.com/2018/07/16/cs-244-18-recreating-and-extending-pensieve/.

[9] Z. Duanmu, K. Zeng, K. Ma, A. Rehman, and Z. Wang. 2017. A Quality-of-Experience Index for Streaming Video. *IEEE Journal of Selected Topics in Signal Processing* 11, 1 (Feb 2017), 154–166. https://doi.org/10.1109/JSTSP.2016.2608329

[10] Sangtae Ha, Injong Rhee, and Lisong Xu. 2008. CUBIC: A New TCP-friendly High-speed TCP Variant. *SIGOPS Oper. Syst. Rev.* 42, 5 (July 2008), 64–74. https://doi.org/10.1145/1400097.1400105

[11] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. 2015. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. *ACM SIGCOMM Computer Communication Review* 44, 4 (2015), 187–198.

[12] Junchen Jiang, Vyas Sekar, and Hui Zhang. 2014. Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive. *IEEE/ACM Transactions on Networking (TON)* 22, 1 (2014), 326–340.

[13] S. Shunmuga Krishnan and Ramesh K. Sitaraman. 2012. Video Stream Quality Impacts Viewer Behavior: Inferring Causality Using Quasi-experimental Designs. In *Proceedings of the 2012 Internet Measurement Conference (IMC '12)*. ACM, New York, NY, USA, 211–224. https://doi.org/10.1145/2398776.2398799

[14] Zhi Li, Xiaoqing Zhu, Joshua Gahm, Rong Pan, Hao Hu, Ali C Begen, and David Oran. 2014. Probe and adapt: Rate adaptation for HTTP video streaming at scale. *IEEE Journal on Selected Areas in Communications* 32, 4 (2014), 719–733.

[15] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. 2017. Neural adaptive video streaming with Pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 197–210.

[16] Michael McCloskey and Neal J Cohen. 1989. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*. Vol. 24. Elsevier, 109–165.

[17] Ricky K. P. Mok, Xiapu Luo, Edmond W. W. Chan, and Rocky K. C. Chang. 2012. QDASH: A QoE-aware DASH System. In *Proceedings of the 3rd Multimedia Systems Conference (MMSys '12)*. ACM, New York, NY, USA, 11–22. https://doi.org/10.1145/2155555.2155558

[18] MPEG-DASH 2012. *Dynamic adaptive streaming over HTTP (DASH) — Part 1: Media presentation description and segment formats*. ISO/IEC 23009-1 (http://standards.iso.org/ittf/PubliclyAvailableStandards).

[19] Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, and Sergey Levine. 2018. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 7559–7566.

[20] Ravi Netravali, Anirudh Sivaraman, Somak Das, Ameesh Goyal, Keith Winstein, James Mickens, and Hari Balakrishnan. 2015. Mahimahi: Accurate Record-and-replay for HTTP. In *Proceedings of the 2015 USENIX Conference on Usenix Annual Technical Conference (USENIX ATC '15)*. USENIX Association, Berkeley, CA, USA, 417–429. http://dl.acm.org/citation.cfm?id=2813767.2813798

[21] X. Nie, Y. Zhao, G. Chen, K. Sui, Y. Chen, D. Pei, M. Zhang, and J. Zhang. 2017. TCP WISE: One initial congestion window is not enough. In *2017 IEEE 36th International Performance Computing and Communications Conference (IPCCC)*. 1–8. https://doi.org/10.1109/PCCC.2017.8280464

[22] Joaquin Quionero-Candela, Masashi Sugiyama, Anton Schwaighofer, and Neil D. Lawrence. 2009. *Dataset Shift in Machine Learning*. The MIT Press.

[23] Anthony Robins. 1993. Catastrophic forgetting in neural networks: the role of rehearsal mechanisms. In *Artificial Neural Networks and Expert Systems, 1993. Proceedings., First New Zealand International Two-Stream Conference on*. IEEE, 65–68.

[24] Anthony Robins. 1995. Catastrophic forgetting, rehearsal and pseudorehearsal. *Connection Science* 7, 2 (1995), 123–146.

[25] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. 2011. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. 627–635.

[26] Kevin Spiteri, Ramesh Sitaraman, and Daniel Sparacio. 2018. From Theory to Practice: Improving Bitrate Adaptation in the DASH Reference Player. In *Proceedings of the 9th ACM Multimedia Systems Conference (MMSys '18)*. ACM, New York, NY, USA, 123–137. https://doi.org/10.1145/3204949.3204953

[27] Kevin Spiteri, Rahul Urgaonkar, and Ramesh K. Sitaraman. 2016. BOLA: Near-optimal bitrate adaptation for online videos. In *INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications, IEEE*. IEEE, 1–9.

[28] Microsoft Research Faculty Summit. 2018. The Good, the Bad, and the Ugly of ML for Networked Systems. (1 August 2018). https://www.microsoft.com/en-us/research/video/the-good-the-bad-and-the-ugly-of-ml-for-networked-systems/.

[29] Yi Sun, Xiaoqi Yin, Junchen Jiang, Vyas Sekar, Fuyuan Lin, Nanshu Wang, Tao Liu, and Bruno Sinopoli. 2016. CS2P: Improving video bitrate selection and adaptation with data-driven throughput prediction. In *Proceedings of the 2016 ACM SIGCOMM Conference*. ACM, 272–285.

[30] Cisco Systems. 2018. Cisco Visual Networking Index: Forecast and Trends, 2017–2022. (26 November 2018). https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.pdf.

[31] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. 2004. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing* 13, 4 (2004), 600–612.

[32] Francis Y. Yan, Jestin Ma, Greg D. Hill, Deepti Raghavan, Riad S. Wahby, Philip Levis, and Keith Winstein. 2018. Pantheon: the training ground for Internet congestion-control research. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. USENIX Association, Boston,

MA, 731–743. https://www.usenix.org/conference/atc18/presentation/yan-francis

[33] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. 2015. A control-theoretic approach for dynamic adaptive video streaming over HTTP. In *ACM SIGCOMM Computer Communication Review*, Vol. 45. ACM, 325–338.