Xatu: Richer Neural Network Based Prediction for Video Streaming

YUN SEONG NAM, Purdue University/ Google, USA
JIANFEI GAO, Purdue University, USA
CHANDAN BOTHRA, Purdue University, USA
EHAB GHABASHNEH, Purdue University, USA
SANJAY RAO, Purdue University, USA
BRUNO RIBEIRO, Purdue University, USA
JIBIN ZHAN, Conviva, USA
HUI ZHANG, Conviva, USA

The performance of Adaptive Bitrate (ABR) algorithms for video streaming depends on accurately predicting the download time of video chunks. Existing prediction approaches (i) assume chunk download times are dominated by network throughput; and (ii) apriori cluster sessions (e.g., based on ISP and CDN) and only learn from sessions in the same cluster. We make three contributions. First, through analysis of data from real-world video streaming sessions, we show (i) apriori clustering prevents learning from related clusters; and (ii) factors such as the Time to First Byte (TTFB) are key components of chunk download times but not easily incorporated into existing prediction approaches. Second, we propose Xatu, a new prediction approach that jointly learns a neural network sequence model with an interpretable automatic session clustering method. Xatu learns clustering rules across all sessions it deems relevant, and models sequences with multiple chunk-dependent features (e.g., TTFB) rather than just throughput. Third, evaluations using the above datasets and emulation experiments show that Xatu significantly improves prediction accuracies by 23.8% relative to CS2P (a state-of-the-art predictor). We show Xatu provides substantial performance benefits when integrated with multiple ABR algorithms including MPC (a well studied ABR algorithm), and FuguABR (a recent algorithm using stochastic control) relative to their default predictors (CS2P and a fully connected neural network respectively). Further, Xatu combined with MPC outperforms Pensieve, an ABR based on deep reinforcement learning.

CCS Concepts: • **Networks → Application layer protocols**; **Network performance modeling**; **Network measurement**.

Additional Key Words and Phrases: Video streaming; Adaptive bitrate algorithms; Predictive models; Neural Networks

ACM Reference Format:
Yun Seong Nam, Jianfei Gao, Chandan Bothra, Ehab Ghabashneh, Sanjay Rao, Bruno Ribeiro, Jibin Zhan, and Hui Zhang. 2021. Xatu: Richer Neural Network Based Prediction for Video Streaming . *Proc. ACM Meas. Anal. Comput. Syst.* 5, 3, Article 44 (December 2021), 26 pages. https://doi.org/10.1145/3491056

Authors' addresses: Yun Seong Nam, Purdue University/ Google, USA; Jianfei Gao, Purdue University, USA; Chandan Bothra, Purdue University, USA; Ehab Ghabashneh, Purdue University, USA; Sanjay Rao, Purdue University, USA; Bruno Ribeiro, Purdue University, USA; Jibin Zhan, Conviva, USA; Hui Zhang, Conviva, USA.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2021 Association for Computing Machinery.
2476-1249/2021/12-ART44 $15.00
https://doi.org/10.1145/3491056

1 INTRODUCTION

Recent years have seen much growth in Internet video, and video traffic is expected to account for 82% of all Internet traffic by 2022 [3]. Despite much progress, consumer surveys indicate that users experience problems such as buffering when streaming video [10]. While access and core network capacity continues to grow, optimizing Internet video delivery will remain a challenge since we are only beginning to see the adoption of technologies such as 4K video [1] which may involve bitrates of tens of Megabits per second, and since there exists (and will likely remain), a wide gulf in the quality of both fixed and mobile broadband connections across households globally [4].

Video streaming today typically involves splitting video into chunks, each encoded at multiple bitrates. Clients pick bitrates for each chunk so as to balance between achieving high video quality, while avoiding rebuffering. Several Adaptive Bitrate (ABR) algorithms [18, 36, 38, 43, 55, 59, 68] have emerged to tackling this challenge. However, the design of these algorithms is complicated by the fact that clients must predict chunk download times, a challenging problem [57].

Most research in video streaming typically assumes that chunk download times are primarily determined by network throughput, and use local algorithms based on observations of prior chunks to estimate throughput [18, 38, 43, 68]. CS2P [57], a state-of-the-art approach, clusters sessions based on features such as the ISP and CDN involved in streaming, and predicts throughput based on a Hidden Markov Model (HMM) trained for each cluster. Yet, predicting throughput is challenging, often leading to overly conservative bitrate selections, or aggressive selections that result in rebuffering.

In this paper, we are motivated by two questions: (i) what information can be made available that can aid in predicting video chunk download times? (ii) how do we design frameworks that can leverage such information to improve the accuracy of prediction for streaming applications? We make the following **contributions**:

First, we present observations from an analysis of a dataset of nearly 100K video session traces from real users. The analysis indicates that (i) while features such as the ISP and CDN aid prediction, pre-clustering sessions based on a combination of these features and learning only from sessions in the same cluster can hurt prediction accuracy; and (ii) chunk download time depends not only on network throughput but also on factors such as the Time to First Byte (TTFB) and chunk size, which are not easy to incorporate into existing prediction approaches [57].

Second, motivated by these observations, we present Xatu[1], a prediction framework for video streaming applications. Xatu considers *static* features that do not vary during a session (e.g., ISP, CDN or city), and models sequences involving multiple *temporal* features that vary across chunks in a session (e.g., TTFB), and not just throughput. The main novel aspects are (i) Xatu learns from other sessions with similar static features, but does not apriori partition data; and (ii) Xatu proposes a gated mask neural network mechanism that uses static features to modify the output of the sequence model, and leverages neural sequence models (specifically, Long Short Term Memory networks (LSTMs) [34]) that can handle multiple temporal features. We show that our output-mask approach gives better accuracy in our task than the traditional approach of combining static and temporal features closer to the input stage of the sequence model used in other domains [27, 53, 62]. We also show that these masks help interpret the role played by static features, providing a naturally automated way of clustering sessions in contrast to pre-defining clusters [57].

Third, we evaluate Xatu using a combination of trace-driven experiments using the above dataset[2], and emulation experiments. Our results show that Xatu reduces the median of the prediction error across sessions relative to CS2P by 23.8%. The benefits come from both the ability of Xatu

[1]Named after a Pokemon character that can see into the future.
[2]We have made the data available at https://github.com/Purdue-ISL/XatuDataset

to exploit static features without pre-clustering data, and from its ability to model chunk-dependent features besides throughput. We integrate Xatu with multiple ABR algorithms including the widely studied MPC [68], and FuguABR [66], a recently proposed approach based on stochastic control which considers perceptual video quality. When combined with MPC, Xatu improves the median of a composite QoE metric [68] across sessions by 38.9% relative to CS2P. The resulting system outperforms Pensieve [43], an ABR algorithm that uses reinforcement learning to make bitrate selections. When combined with FuguABR, Xatu achieves significantly lower rebuffering (the 80^{th} rebuffering ratio is lowered from 10% to 0%), and higher QoE scores relative to FuguNN, a fully connected neural network proposed in [66].

We evaluate Xatu on an additional dataset obtained using controlled measurements, which also has available the CDN layer an object is served from. The results further confirm Xatu's benefits, and show that Xatu is extensible, and can easily exploit new features such as the CDN layer from which a chunk is served to further improve prediction accuracies.

2 BACKGROUND

In HTTP-based streaming, videos are split into chunks (corresponding to a few seconds of playtime), with each chunk encoded at multiple bitrates (corresponding to different qualities). A video player selects a bitrate level for each chunk taking into account the amount of data already buffered locally, and a prediction of how long downloading a chunk at a given bitrate would take. In doing so the player seeks to balance multiple video delivery metrics. These metrics include (i) the average bitrate across chunks; (ii) the rebuffering ratio, i.e., the fraction of the video session for which the video player encounters rebuffering; (iii) the extent to which bitrate levels change in the video; and (iv) the join time. A variety of ABR algorithms have been developed to make these decisions[18, 36, 38, 43, 55, 68], which primarily differ based on how the algorithms choose which bitrate to use for each chunk by reconciling the above factors. While some ABR algorithms only take client buffer occupancy into account [36, 55], the vast majority of algorithms rely on predictions of chunk download times. We discuss factors that impact chunk download times and current algorithms.

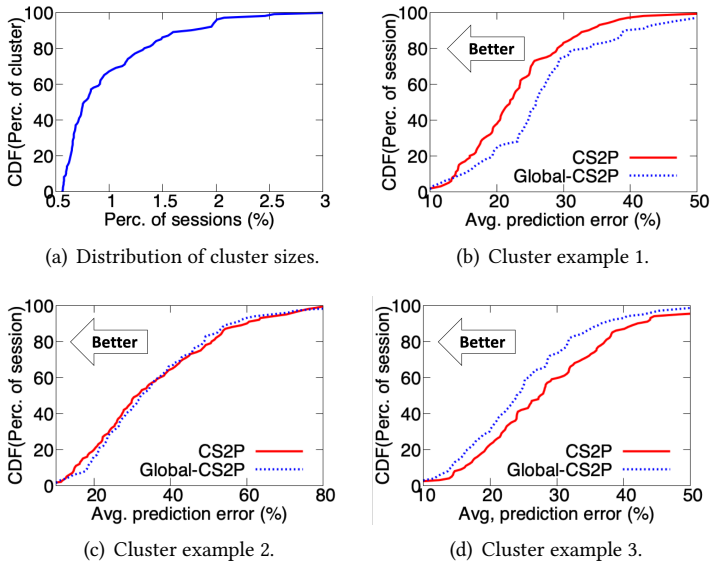
Many factors impact chunk download times. The download time D_i of chunk i refers to the time from when a HTTP request is sent for the chunk to when download is finished. We have:

$$D_i = \text{TTFB}_i + \frac{S_i}{\text{BW}_i} \tag{1}$$

Here, TTFB_i represents the Time To First Byte (TTFB) (i.e., the time from when a HTTP request for a chunk is made to when the first byte of a response is received) for chunk i, S_i is the size of the chunk, and BW_i is the network (TCP) throughput. Further, $\frac{S_i}{\text{BW}_i}$ represents the receive time (i.e., the time from when the first byte of the response is received to the last byte). We refer to the ratio of the size of a chunk to its download time as the **chunk download rate (CDR),** which for chunk i is:

$$CDR_i = \frac{S_i}{D_i} = \frac{S_i}{\text{TTFB}_i + \frac{S_i}{\text{BW}_i}} \tag{2}$$

ABR algorithms primarily focus on network throughput. Most works in the video streaming literature (e.g., [18, 38, 68]) do not explicitly consider TTFB, and do not distinguish between chunk download rates and network throughput. Effectively, these works assume the TTFB is small, in which case the chunk download rate is the same as the network throughput, and the impact of chunk size is not considered. A common approach, first proposed in [38], and later adopted by others [18, 68], predicts the download rate of a future chunk based on the harmonic mean of the



(a) Distribution of cluster sizes.
(b) Cluster example 1.
(c) Cluster example 2.
(d) Cluster example 3.

Fig. 1. (a) cluster size distribution, and (b)-(d) are the prediction error of chunk download rates with CS2P and Global-CS2P for example clusters.

download rate of prior chunks. More recently, CS2P [57] has explored the use of a Hidden Markov Model (HMM) for predictions, based on the observation that TCP throughput within a session exhibits stateful characteristics and depends on hidden states (e.g., number of flows sharing a bottleneck link).

The state-of-the-art approach clusters sessions to aid prediction. CS2P [57] also observed that sessions sharing a combination of similar features such as ISP, CDN, user location, and time-of-day tend to have similar network characteristics. Based on these observations, CS2P (i) clusters sessions based on the above features; and (ii) trains a HMM model for each combination of features, rather than using a generic HMM trained across all sessions.

3 MOTIVATING DATA ANALYSIS

In this section, we analyze a dataset of real video streaming sessions to understand the implications for prediction approaches used in ABR algorithms today.

Datasets. Our dataset comprises of approximately 100,000 video session traces from real users collected over the course of three months in 2017 from a publisher in the United States. For each video session trace, and for each chunk, the data includes information such as the (i) chunk size; (ii) TTFB and (iii) the start and end times of the download. Each trace also has anonymized IDs representing each of the CDN and ISP that served the video session, an anonymized ID representing the geo-location of user (*e.g.* city and country), and the time-of-day when the video session started. Video sessions in our dataset are served by 2 CDNs and 89 ISPs across 1406 cities. The average chunk download rate of sessions varies from 1.31 Kbps to 94.89 Mbps.

We analyze the data with a view to analyzing two questions (i) does clustering data in a fashion similar to CS2P [57] improve prediction accuracy; and (ii) the extent to which the TTFB impacts chunk download times.

3.1 Impact of clustering

We clustered video sessions based on features which include ISP, CDN, city and time-of-day (we use 6 hour windows starting from midnight which is in the range suggested in [57]), and trained a HMM for the specific combination of features as done in CS2P [57].

After clustering data as above, we found many clusters with fewer than 150 sessions, which we filtered as suggested by [57]. After the filtering, we were left with approximately 100 clusters with up to 800 video sessions in each cluster, and a total of 27,000 video traces. Figure 1(a) shows a CDF of the percentage of sessions that falls inside each of the remaining clusters. The clusters range in size from about 0.5% of all sessions to about 3% of the sessions. Since CS2P [57] uses each cluster separately in its training, the apriori clustering approach implies that only a small portion of the data is available for the training in any given cluster.

For sessions in each cluster, we compare the error in predicting chunk download rate using a HMM trained at each cluster (which we refer to as *CS2P*), against that of using a HMM trained over all sessions across all clusters (which we refer to as *Global-CS2P*). We present training methodology details in §5.1, and report the prediction error in chunk download rates, averaged across chunks (more formally defined in §5.3). Figure 1 compares the prediction error of *CS2P* against *Global-CS2P* for three example clusters that contain 0.59%, 1.14% and 0.64% of total sessions respectively. In each graph, a curve to the left corresponds to lower error. While clustering is beneficial in some cases (Figure 1(b)), it does not always provide benefits (Figure 1(c)) and can even hurt (Figure 1(d)).

More generally, we find that clustering helps if the sessions within the cluster have sufficiently similar network characteristics, where limiting the HMM to these similar sessions is beneficial. However, if sessions within a cluster exhibit disparate network characteristics, the benefits of clustering must be weighed against the fact that the overall training data is much smaller. Further, pre-clustering data prevents learning from sessions in other related clusters. For instance, it may be beneficial to learn from sessions in clusters corresponding to slightly different times of day, or nearby cities since these sessions may have similar network patterns. We explore this further in §5.3.

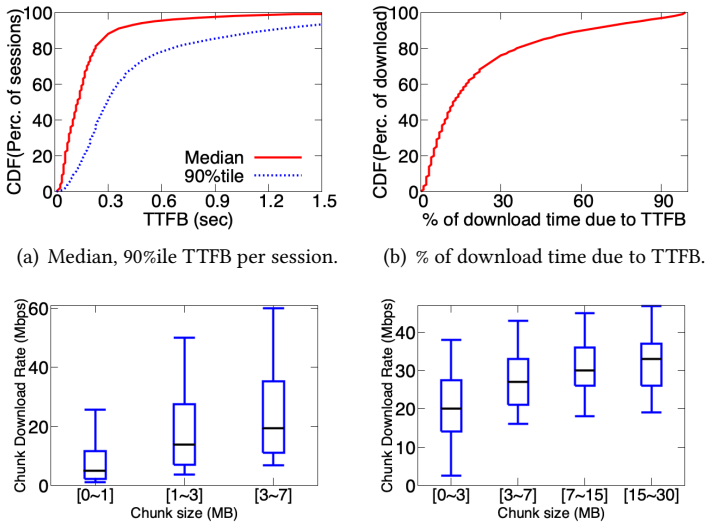
3.2 Impact of TTFB

To understand the impact of TTFB on chunk download times, we analyze the original dataset but only consider sessions corresponding to the country ID with the most sessions.

TTFB can significantly impact chunk download times. For each video session, we consider the median TTFB and 90%ile TTFB obtained across all chunks downloaded in that session. Figure 2(a) shows the distribution of the median and 90%ile TTFB across the sessions. While the median TTFB for most sessions is small, 23% of the sessions have a median TTFB that exceeds 200 milliseconds, even exceeding 1 second in a few cases. Further, the 90%ile TTFB of most sessions is high exceeding 300 milliseconds for half the sessions, and exceeding 1 second for nearly 15% of the sessions.

Figure 2(b) shows the percentage of the chunk download time that may be attributed to the TTFB across all video chunks of all sessions. For 40% of the chunk downloads, more than 20% of the download time may be attributed to TTFB, while for 15% of chunks downloads, more than half the download time is attributable to the TTFB. These results indicate that TTFB is a significant contributor to chunk download time.

While it is difficult to definitely say why the TTFB is so significant given the information available in our dataset, we note that the TTFB depends on both the latency from the client to the CDN edge server from which the chunk is downloaded, as well as the time taken by the CDN edge server to retrieve the chunk on a cache miss (from either a remote CDN server or the origin). In §7, we confirm our finding about TTFB from a separate dataset collected from various video publishers



(a) Median, 90%ile TTFB per session.

(b) % of download time due to TTFB.

(c) Distribution of chunk download rate across chunks for different size ranges from our datasets.

(d) Distribution of chunk download rate across chunks for different size ranges from a controlled experiment.

Fig. 2. Impact of TTFB and chunk size on chunk download time and rate.

through controlled experiments. We also discuss how potential variations in TTFB and chunk download rates may arise based on where in the CDN hierarchy a chunk is served from, and some implications for predicting chunk download rate.

Chunk download rate depends on chunk size. One consequence of TTFB being an important factor that impacts download times is that chunk download rates depend on chunk sizes (as follows from Equation 2). Figure 2(c) verifies this by showing the chunk download rate observed by the clients across all chunks in video sessions. The chunks are categorized by size into different ranges, and each boxplot shows the distribution of chunk download rates for each chunk size range. The box corresponds to the 25^{th} and 75^{th} percentiles, the horizontal bar to the median, and the end points of the vertical line correspond to the 10^{th} and 90^{th} percentiles. The figure clearly shows that the chunk download rate tends to be higher for larger chunk size.

An artifact of our dataset that could cause the dependence on chunk size is that it is collected using video sessions that run ABR algorithms in the wild, and these algorithms may download larger bitrates (and hence larger chunk sizes) when network conditions are better. To ensure this is not the primary factor that causes the dependence of chunk download rates on chunk sizes, we collected a separate dataset using controlled experiments (detail can be found in §7) that ran video sessions with other publishers from a home network, and after disabling the ABR algorithm and using the highest bit rate for all chunks. This still results in variability in chunk sizes owing to the variable bitrate encoding. Figure 2(d) shows boxplots of the chunk download rates observed using these controlled experiments. The result confirms the above trend, and shows that even when the ABR algorithm is disabled, chunk download rates are larger for larger chunk sizes.

HMMs are restricted in terms of which features can be modeled. Explicitly modeling the impact of TTFB and chunk size on download times is not straightforward with HMMs. One approach is to build a separate HMM for each range of chunk sizes. However, this approach is more appropriate if sizes stay within the range throughout the session. Since a HMM models the network

as having a sequence of state transitions, and given that the size itself might change during the session, it is not clear how to design a model with multiple HMMs that interact.

Another option is to model network state as a multivariate distribution whose output is the tuple that comprises both the TTFB and the network throughput. This multivariate HMM jointly predicts the TTFB and network throughput for any given network state. For a given chunk size, the download time is appropriately predicted from the predicted TTFB and network throughput, at least in theory. However, when a HMM is trained for this task, it minimizes the prediction error of the tuple TTFB and network throughout, without caring how these prediction errors impact the final download time prediction. Directly predicting the final download time is difficult in HMMs, as we need to explicitly model the inverse distribution of TTFB_i and BW_i from Equation (1), which does not have a closed form expression for typical assumptions about the distributions of TTFB_i and BW_i. In fact, we did try a multivariate HMM approach and found it led to significantly higher prediction error. Specifically, the average prediction error for the median session with the multivariate HMM was 72.4% in contrast to CS2P using a single HMM based on chunk download rate which was 43.4%.

4 XATU DESIGN

In this section, we present Xatu, which is motivated by the observations in §3 and a few key ideas:

Learn from relevant sessions without apriori clustering and data pre-partitioning. Xatu uses features such as the ISP, CDN, time-of-day and city to aid predictions. We henceforth refer to such features as *static features* since they do not change during the session. [3] However, rather than apriori clustering data based on static features (as done by CS2P in §3.1), Xatu uses these features as inputs, and learns across all sessions. This allows Xatu to automatically learn from sessions with related but not identical features (e.g., sessions with different cities, or different values of time-of-day that may yet have similar network characteristics).

Model sequences with multiple chunk-dependent features (e.g. TTFB), not just throughput. Unlike existing ABR prediction approaches [38, 57], Xatu explicitly models TTFB and accounts for the fact that chunk download rates do not necessarily coincide with throughput. Specifically, Xatu models multiple features associated with each prior chunk in the session including the TTFB, chunk size, network throughput, and download time. We henceforth refer to these features as *temporal features*, since they change across chunks within a session.

Effectively combining static and multiple temporal features. A key component of Xatu is how to effectively combine both static and temporal features. To achieve this, we have developed a neural network architecture customized to this purpose rather than use an off-the-shelf approach for reasons that we discuss further in §4.1. In §5.3, we show the importance of considering static features, as well as the benefits of Xatu's custom neural network.

We discuss Xatu's architecture in §4.1, and present design details in §4.2.

4.1 Xatu Architecture

Consider a video session with n static features, denoted $s_1^{(j)}, \ldots s_n^{(j)}$, where each feature can take any of an appropriate set of discrete values. Likewise, we use $d_{i,t}^{(j)}$ to denote temporal feature i (TTFB, size, etc.) of chunk t of video session j. We use $d_t^{(j)} = (d_{1,t}^{(j)}, \cdots d_{m,t}^{(j)})$ to refer to an m-dimensional vector that contains all temporal features associated with the t-th chunk of video session j.

[3] Xatu currently models the CDN as a static feature since switching of CDNs during a session occurs relatively infrequently, but it is easy to move the CDN to a temporal feature if beneficial.

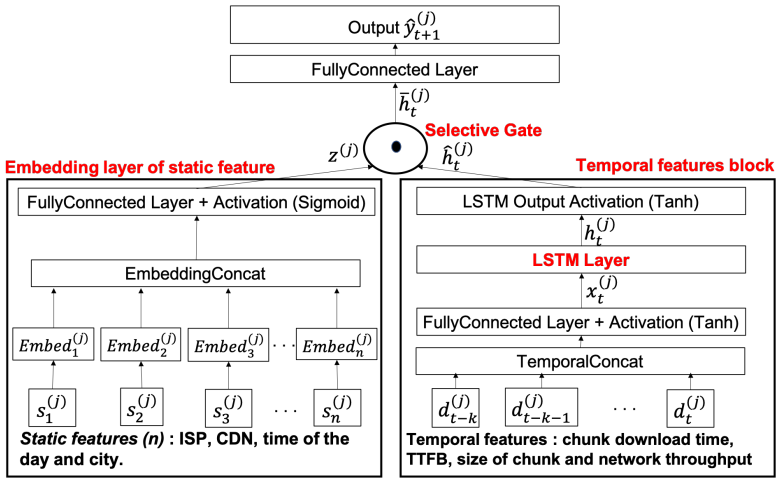


Fig. 3. Xatu architecture.

Xatu uses a Long Short Term Memory network (LSTM) [34], which is a special type of Recurrent Neural Networks (RNN) capable of learning long-term dependencies. LSTMs are sequence models that can be used to predict the next value (or next few values) in a time series. Second, LSTMs can handle multivariate inputs, automatically finding complex (including possibly non-linear) relationships between inputs and outputs. LSTMs have been used successfully in various contexts including language modeling, text classification, time series modeling, and anomaly detection.

In architecting Xatu, an important question is how best to combine the static and temporal features. A conventional approach, used in question and answer systems [37, 42, 63], is to use static features in the input, as in the architecture shown in Figure 6(a) where the static and temporal features are concatenated at the input stage. However, a key issue with directly adapting this approach to our context is that the LSTM is not directly aware of which part of the information stays the same and it may take multiple samples to learn the invariant portions of the inputs.

Rather, Xatu introduces an alternate approach to combine static and temporal features, shown in Figure 3. The architecture comprises separate static and temporal feature blocks. We use the static features to generate a gate mask: each neuron of the LSTM output (output related to the temporal features) can be turned off depending on the static features. The gate mask directly encodes the effect of static features in the model predictions.

In more detail, through the *selective gate* (Figure 3), the static feature attention vector ($z^{(j)}$) selects which of the dimensions of the temporal embedding $\hat{h}_t^{(j)}$ are relevant to the task. This allows us to see which static features force the model to pay attention to the same dimensions of the temporal embedding $\hat{h}_t^{(j)}$, changing how the model behaves with different time series. These will be deemed similar static features, since they lead to similar model outputs with respect to the time series inputs. The best analogy of the *selective gate* is to think of these embeddings as bit vectors. The bit vectors of the static feature embedding act as a mask over the temporal series embedding. If sessions j and i have $z^{(j)} \approx z^{(i)}$, then their Xatu models will have similar behavior if given similar temporal sequences. We note that the same $z^{(j)}$ applies to all $\hat{h}_t^{(j)}$ regardless of the timestep t. In practice, the gate mask is implemented to assume values in the range $(0, 1)$, to ensure that the static input model is differentiable.

To our knowledge, the closest work to our gated approach is the tangentially related use of attention for position encoding in natural language processing tasks [70]. We empirically evaluate

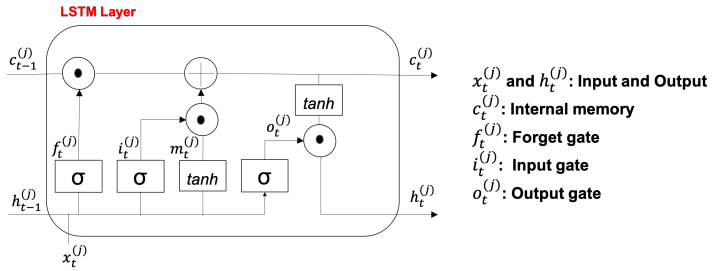


Fig. 4. Detail of LSTM layer.

the benefits of our approach (Figure 3) over the architecture in Figure 6(a) in §5.3. Another alternative to our approach is to concatenate the output of the static and temporal feature blocks in Figure 3 before going through a fully connected layer. We did not adopt this approach since the static and temporal features interact in ways that are not easy to interpret. Instead, Xatu's approach allows us to interpret the impact of static features as illustrated in §5.3.

4.2 Design details

We discuss the key building blocks below:

Embedding layer of static features. Figure 3 (left) shows the embedding block that takes the static features of session j and outputs the mask $z^{(j)}$. It starts with Xatu assigning one-hot encodings to all our n categorical features (e.g., ISP name, CDN name, city, time-of-day, etc.), denoted $s_1^{(j)}, \ldots, s_n^{(j)}$ for session j. We classify the time-of-day into multiple bins (4 bins of 6 hours each in our evaluations), and treat it as a categorical feature because of its cyclic nature. These n inputs are then each passed through n multilayer perceptrons (MLPs), fully connected linear layers with non-polynomial activation functions, $\text{Embed}_1^{(j)}, \ldots, \text{Embed}_n^{(j)}$; there are n distinct MLPs, one for each input. The output of these MLPs are then concatenated into a single output and passed through another MLP with sigmoid activations (as many output neurons as there are neurons in the $h_t^{(j)}$ output (Figure 4) of the LSTM), resulting in a final output mask $z^{(j)}$. This mask will be latter elementwise multiplied with $\hat{h}_t^{(j)}$ in the selective gate to ensure the throughput prediction is influenced by the static features of session j.

Temporal features block. Here, we consider predicting the download time of chunk $t+1$ at session j from the sequence $d_1^{(j)}, d_2^{(j)}, \ldots, d_t^{(j)}$ —as defined earlier— and the desired chunk size at time $t+1$. Rather than only considering the features of chunk t, Xatu considers the features of the past k chunks of the session, where k is a hyperparameter (k is referred to as an *input frame*, and we typically use $k = 5$). So the input of LSTM is the concatenation of temporal features from the last k chunks which results in an $m \times k$ matrix $\text{CONCAT}(d_{t-k}^{(j)}, \ldots, d_t^{(j)})$. Even though in theory an LSTM can learn both long and short term historical dependencies of inputs, explicitly providing the last k chunks ensures it pays close attention to recent history and avoids optimization challenges with backpropagation-through-time (such as vanishing and exploding gradients). Doing so rather than just providing the last chunk as input improves prediction accuracy. Another input is the chunk size that has been requested for time $t+1$.

These inputs are fed into an MLP with hyperbolic tangent (tanh) activations outputting vector $x_t^{(j)}$. For completeness, we also show the schematic of the LSTM in Figure 4. Let $c_t^{(j)}$ denote the internal memory of the LSTM layer at the end of time step t (in our context, this corresponds to chunk t of a given video session, with the memory being set to null state at the start of the session). Given an input $x_t^{(j)}$ at time step t, the LSTM combines this input with the memory of the previous

step ($c_{t-1}^{(j)}$), and the previous output ($h_{t-1}^{(j)}$) to produce an output $h_t^{(j)}$, and updates its memory to $c_t^{(j)}$. In doing so, the LSTM uses multiple gates as described below.

The forget gate layer (with output $f_t^{(j)}$) determines which information from the previous memory is retained and discarded by the LSTM layer. The input gate layer (output $i_t^{(j)}$) decides which values in the memory are updated, while the tanh layer produces $m_t^{(j)}$ which determines new information to be added to the memory. The memory is updated by combining $i_t^{(j)}$ and $m_t^{(j)}$. The output $h_t^{(j)}$ is based on combining the memory (after it goes through a tanh layer), with $o_t^{(j)}$ (the result of the output gate which determines which parts of the memory to output). Finally, the temporal features block outputs $\hat{h}_t^{(j)}$ at download of chunk t after the activation.

Combining output of static and temporal blocks. The final output combines $z^{(j)}$ — which is a function of the static session features $s_1^{(j)}, \ldots, s_n^{(j)}$ — with the output of temporal features block $\hat{h}_t^{(j)}$, resulting in the download time prediction $\hat{y}_{t+1}^{(j)} = \mathbf{w}_{\text{final}}^T \overline{h}_t^{(j)}$, where $\mathbf{w}_{\text{final}}$ is a vector of parameters of the same size as $\overline{h}_t$ and $\overline{h}_t^{(j)} = z^{(j)} \odot \hat{h}_t^{(j)}$ is the Hadamard product ($\odot$) –element-wise multiplication– between the mask of the static features $z^{(j)}$ and the output of temporal features block $\hat{h}_t^{(j)}$.

Training Xatu. Our whole framework is simply a function

$$\hat{y}_{t+1}^{(j)}, c_t^{(j)}, h_t^{(j)} = F\left(\mathbf{s}^{(j)}, d_{t-k}^{(j)}, \ldots, d_t^{(j)}, b_{t+1}^{(j)}, c_{t-1}^{(j)}, h_{t-1}^{(j)}, \mathbf{W}\right),$$

where $\mathbf{s}^{(j)}$ is the static session features $s_1^{(j)}, \ldots, s_n^{(j)}$, $b_{t+1}^{(j)}$ is the chunk size requested for time $t+1$, $\hat{y}_{t+1}^{(j)}$ is a prediction of $t+1^{\text{st}}$ chunk download time, and $\mathbf{W}$ refers to the set of neural network parameters in all the layers.

To find parameters $\mathbf{W}$ by training Xatu, we first define a loss function to evaluate a prediction accuracy. A loss function L is an sum (or a average) of prediction error l of chunk t

$$L\left(y_1^{(j)}, \cdots, y_{C^{(j)}}^{(j)}, \hat{y}_1^{(j)}, \cdots, \hat{y}_{C^{(j)}}^{(j)} | \mathbf{W}\right) = \sum_{t=1}^{C^{(j)}} l\left(y_t^{(j)}, \hat{y}_t^{(j)}; \mathbf{W}\right),$$

where $C^{(j)}$ is a total number of chunks downloaded in session j, and $y_t^{(j)}$ and $\hat{y}_t^{(j)}$ are the actual and predicted t-th chunk download times of session j, respectively. The prediction error $l\left(y_t^{(j)}, \hat{y}_t^{(j)}\right)$ can be any form of an error metric such as a square error or an absolute error between a predicted download time and an observed download time. Then we update parameters $\mathbf{W}$ by gradient descent, calculating the loss gradient as

$$\frac{\partial L}{\partial \mathbf{W}} = \sum_{t=1}^{C^{(j)}} \frac{\partial}{\partial \mathbf{W}} l\left(y_t^{(j)}, \hat{y}_t^{(j)}; \mathbf{W}\right).$$

The overall optimization samples a session j without replacement from the dataset. Once all sessions are exhausted, an epoch has ended, and we start again by sampling sessions without replacement from the dataset. Because $h_t^{(j)}$ affects all the losses $\left\{l\left(y_{t'}^{(j)}, \hat{y}_{t'}^{(j)}; \mathbf{W}\right)\right\}_{t'>t}$ the computation of the gradient requires a technique called backpropagation through time (BPTT). However, if the length of video session j, $C^{(j)}$, is very long, the gradient can either diverge (explode) or vanish (become zero). To avoid these optimization issues, we split a video session into shorter segments. Note that each BPTT block will use the last hidden state (for both $c_t^{(j)}$ and $h_t^{(j)}$) from its previous BPTT block as an initial state, only it will not propagate the gradient back. At the start of a session, variables $c_1^{(j)}$ and $h_1^{(j)}$ are both set to zero.

5 EVALUATION: PREDICTION ACCURACY WITH XATU

In this section, we evaluate (i) the effectiveness of Xatu in improving prediction accuracy relative to CS2P [57] (the state-of-the-art prediction approach for ABR streaming); and (ii) the benefits of Xatu's architecture. In §6, we evaluate how these benefits translate to better performance for ABR algorithms. We begin with implementation details of Xatu, and CS2P [57], and also discuss datasets and training methodology.

5.1 Implementation detail

We implemented Xatu and CS2P as follows:

Xatu. We implement Xatu using the PyTorch package [13]. We used a BPTT size of ten (chunks) and an input frame size (k) of five in our implementation. We arrived at these parameters after tuning experiments where we varied each of the parameters between two and twenty, and chose the settings that resulted in the best accuracy in our validation dataset (though we note that the performance was comparable for a wide range of parameter choices). Based also on a hyper-parameter search on the same validations data, we used 516 hidden units for each layer in Xatu. By default, the sum of absolute prediction error in each BPTT, loss $= \Sigma_{p=0}^{B} \mid y_{t-p}^{(j)} - \hat{y}_{t-p}^{(j)} \mid$, where B is the BPTT size. We have also considered other loss functions, as we discuss in 5.3.

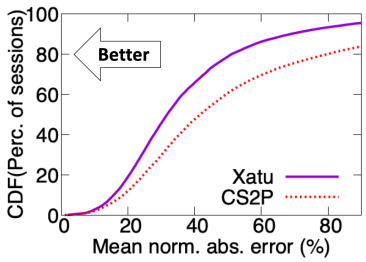
By default, Xatu uses the following static features: (i) ISP; (ii) CDN; (iii) time-of-day and (iv) city. Further it uses the following temporal features for each chunk i: (i) TTFB ($TTFB_i$); (ii) chunk size (S_i); (iii) network throughput (BW_i) and (iv) download time (D_i). $TTFB_i$, S_i and D_i are directly measured, while BW_i is derived from Equation 1 since all other parameters of that equation are known. Measuring $TTFB_i$ is not difficult. For instance, in our experiments in §6 with the dash.js player, $TTFB_i$ was obtained using timestamps provided by the player measured when the request was sent, and the first byte was received. We focused on these features given availability in our datasets. However, unlike HMMs, Xatu's architecture is general and can accommodate additional static and temporal features easily. For instance in §7, we collected an additional smaller dataset to instrument where in the CDN hierarchy an object is served from and augmented Xatu to leverage this additional temporal feature.

CS2P. We implemented CS2P using the hmmlearn python package [9]. We use the dataset described in §3 into clusters using the features described in §3 and build a separate HMM for each cluster following [57]. The performance of CS2P depends on the number of hidden states (N), We vary N between 2 and 20 as suggested by [57], and pick the best choice of N for each cluster that results in the lowest average prediction error for that cluster on the validation data.

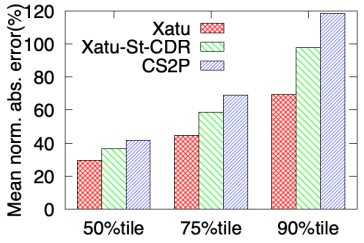
5.2 Datasets and training

Our evaluations were mainly conducted using data collected from real-world video streaming sessions(§3). Specifically, we use the set of 27K traces described in §3.1 which we henceforth refer to as the *Primary DataSet*. Recall that this set was obtained after clustering sessions based on a combination of static features and filtering out traces belonging to small clusters(§3.1). We use the *Primary DataSet* to ensure that the performance of CS2P was not negatively impacted by the presence of traces from clusters that were too small to allow per-cluster HMM training.

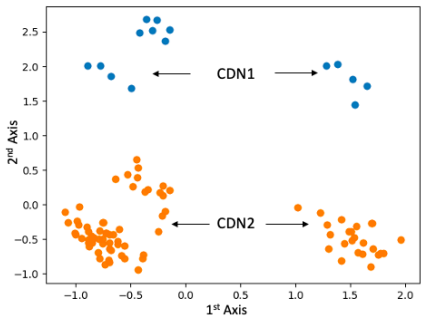
For CS2P, we clustered data using the combination of the features mentioned above. For each cluster, we used 60% of the traces for training a HMM for that cluster, 20% for validation, and 20% for testing. Since Xatu does not need a separate model for each cluster, we merged the training sets from all clusters to train Xatu (we used a learning rate of 0.01). Likewise, when we evaluate Global-CS2P which involves a single HMM across all traces, we used the same merged training set



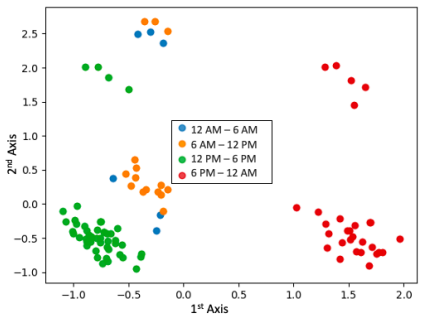
(a) Mean NAE(%) with Xatu and CS2P in the validation set of the *Primary DataSet*.



(b) Understanding how Xatu obtains its benefits.



(c) 2D projection of $z^{(j)}$ for each CDN cluster through PCA.



(d) 2D projection of $z^{(j)}$ for each time of day (ToD) cluster through PCA.

Fig. 5. Prediction errors with various schemes and understanding Xatu's approach.

across all clusters. Our experiments related to prediction accuracy report results for the validation set. The traces in the testing set were used for our testbed emulation experiments (§6).

5.3 Results

We compare the prediction error in chunk download rates[4] by considering each session, taking the normalized absolute error (NAE) per chunk and computing the mean across chunks. Formally, if $C^{(j)}$ is the number of chunks downloaded, and $r_t^{(j)}$ and $\hat{r}_t^{(j)}$ are the predicted and actual chunk download rates for chunk t for session j, we report:

$$\frac{1}{C^{(j)}} \sum_{t=1}^{C^{(j)}} \left| \frac{r_t^{(j)} - \hat{r}_t^{(j)}}{r_t^{(j)}} \right|,$$

Xatu vs CS2P. Figure 5(a) shows the CDF of the mean NAE across sessions for Xatu and CS2P in the validation set of the *Primary DataSet* (§5.2). Xatu reduces the median and 90%ile of the mean NAE across sessions by 23.8% and 41.8% respectively. We have considered many other metrics to summarize prediction error, and find Xatu consistently out-performs CS2P. For instance, Xatu reduces the median NAE across chunks in a session by 10.3% for the median session, and the 90%ile NAE across chunks by 20% for the median session. We present more detailed results in the Appendix.

Factors that help Xatu's performance. There are two factors that help Xatu compared to CS2P. First, rather than pre-partitioning data into clusters, Xatu jointly learns from all data but

[4]Given a chunk of a given size, Xatu predicts download time, which we convert into a chunk download rate using Equation 2.

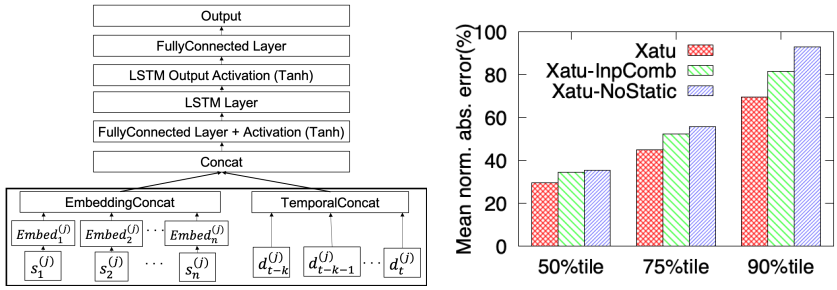
uses the static features to learn from relevant sessions. Second, Xatu models multiple temporal features that determine chunk download time including TTFB. Recall that temporal features such as TTFB are not easily incorporated with CS2P's HMM model. To separate these benefits, we consider a variant of Xatu which uses all static features but considers chunk download rate as the only temporal feature (TTFB is not modeled), exactly the information CS2P uses. We refer to this model as *Xatu-St-CDR*.

Figure 5(b) shows a bar graph which presents the median, $75^{th}\%ile$ and $90^{th}\%ile$ of the mean NAE across sessions for Xatu, Xatu-St-CDR and CS2P. Across all percentiles, Xatu-St-CDR achieves lower mean NAE than CS2P, and these benefits may be attributed to Xatu learning across all relevant sessions rather than only learning from sessions in the same cluster. Likewise Xatu achieves lower mean NAE across all percentiles compared to Xatu-St-CDR, which may be attributed to Xatu modeling TTFB, while Xatu-St-CDR only considers chunk download rate. This result also shows the benefits of considering multiple temporal features with Xatu rather than CDR alone.

Interpreting how Xatu uses static features. A key aspect of the Xatu architecture is that it facilitates interpretation of how it uses the static features (§4.1). We consider the gate mask, $z^{(j)}$ (§4.1, Figure 3)) associated with each distinct combination of static features. Figure 5(c) and Figure 5(d) show results obtained by using Principal Component Analysis (PCA) [12] to project $z^{(j)}$ (a vector of 516 dimensions in our experiments) into a 2 dimensional space to see whether gate masks for different static feature combinations are similar (each dot corresponds to a different combination which is referred to as a cluster by CS2P). Further, Figure 5(c) colors each static feature combination by the CDN. Figure 5(d) is identical except that each static feature combination is colored by the time of day (ToD). We make several points. First, Figure 5(c) shows that combinations with the same CDN are clearly separated with the top three groups from one CDN (orange dots), and the bottom three groups corresponding to another CDN (blue dots). Recall that our dataset has 2 CDNs. Second, Figure 5(d) shows that the combinations corresponding to each CDN are clearly separated by ToD. The combination on the right (red dots) are clearly separated from the other clusters and correspond to the 6pm-12am block. Further, the orange and blue dots are relatively close, corresponding to the 12am-6am and 6am-12pm blocks respectively. In the Appendix, we have presented the same information, but coloring each combination with the ISP (15(a)), and city (15(b)). The figures do exhibit some interesting patterns in terms of how these features further separate combinations with the same CDN and ToD, but overall the results suggest that the greatest variation across models occurs based on the CDN and ToD for which Xatu found it important to have specialized predictions. This points to the advantage of Xatu's neural network architecture (§4.1) which facilitates interpretation of how static features are used. Note also that Xatu's approach has a benefit over pre-clustering since it can learn from models developed with combinations with overlapping features - for instance, in the example, if Xatu encounters a session from a city and ISP for the first time, it could still leverage information from other sessions with the same CDN and ToD.

Benefits of static features and Xatu's NN architecture. To highlight the importance of considering static features in Xatu, we consider a variant of Xatu which only uses temporal features, and completely disables all static features. We refer to this variant as Xatu-NoStatic.

Further, Xatu not only considers static features, but also uses a custom neural network architecture that combines static and temporal features at the output with each unique set of static features resulting in a unique gate mask (§4.1). We compare this architecture of Xatu with a more traditional alternate architecture (Figure 6(a)) used in other domains which we refer to as Xatu-InpComb that combines the static and temporal features at the input and does not involve a selective gate. We perform an ablation study comparing these different designs.



(a) Alternative architecture to Xatu combining static features at the input.

(b) Benefits of Xatu's architecture

Fig. 6. (a)-(b) an ablation study of Xatu architecture

Figure 6(b) shows a bar graph which presents the median, $75^{th}\%ile$ and $90^{th}\%ile$ of the mean NAE across sessions for Xatu, Xatu-InpComb and Xatu-NoStatic. We observe: (i) Across all percentiles, Xatu achieves lower mean NAE than Xatu-NoStatic indicating the importance of considering static features. ii) Xatu achieves lower mean NAE compared to Xatu-InpComb showing the benefits of Xatu's approach in combining static and dynamic features. In fact, Xatu-InpComb only shows marginal improvement over Xatu-NoStatic indicating that it is not only important to consider static features but also important to carefully design the neural network to achieve prediction benefits.

We have also evaluated the benefits of using the last k chunks in Xatu rather than just the last one more common with LSTMs. We found that when using $k = 5$ (chosen based on hyper-parameter tuning), the median of the mean Normalised Absolute Error (NAE) across sessions reduces to 31.9% from 34.3%, with similar reductions in other percentiles of mean NAE as well.

Sensitivity to loss function. Xatu can use any form of loss function during the training (§5.1). To check the impact of various loss functions on throughput prediction accuracy of Xatu, we tried four different loss functions; (i) $loss1 = \Sigma_{p=0}^{B} \mid y_{t-p}^{(j)} - \hat{y}_{t-p}^{(j)} \mid$, (ii) $loss2 = loss1/C^{(j)}$, (iii) $loss3 = \Sigma_{p=0}^{B} \left(y_{t-p}^{(j)} - \hat{y}_{t-p}^{(j)} \right)^2$, and (iv) $loss4 = loss3/C^{(j)}$, where B and $C^{(j)}$ are a size of BPTT and length of session j respectively, and $\hat{y}_t^{(j)}$ and $y_t^{(j)}$ are a predicted and an actual throughput at download of chunk t at session j respectively. While loss1 and loss3 use sum of absolute or square error, loss2 and loss4 divide the sum by a length of a session ($C^{(j)}$ of session j). Figure 7 shows the distribution of mean NAE with Xatu across sessions for above loss functions. We conclude that even when the evaluation criteria is NAE, the performance across loss functions shows similar throughput prediction accuracy, pointing to the robustness of Xatu w.r.t. the chosen loss objective.

6 EVALUATION: BENEFITS ON ABR ALGORITHMS

To show the benefits of Xatu in video delivery performance, we have integrated Xatu and CS2P with MPC, a widely used ABR algorithm [68], replacing its default harmonic mean throughput predictor. We also integrate Xatu with the ABR algorithm proposed recently in Fugu [66], and compare its performance relative to the default fully connected neural network (§6.1).

Evaluation Testbed. We create an emulation setup involving video being streamed to a Dash.js video player on Chrome [6] from an Apache server hosting the video. We emulate traces from the testing set above (§5.2), and vary both the network throughput and the TTFB as per the trace.

Like previous works [18, 43, 57], we modify the Dash.js video player so requests for which bitrate to select are sent to a server (which we refer to as an ABR server). The server queries the appropriate prediction model (CS2P, or Xatu) and picks the bitrate chosen by MPC for that prediction. Likewise,

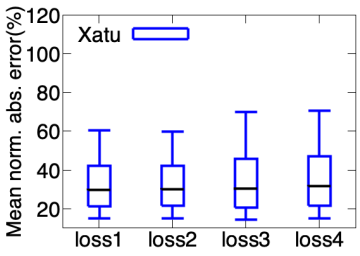


Fig. 7. Sensitivity of Xatu to loss functions.

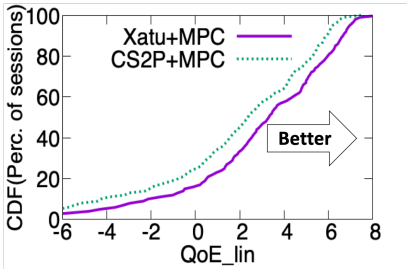


Fig. 8. QoE-lin of Xatu+MPC and CS2P+MPC from the testbed emulation on the testing set.

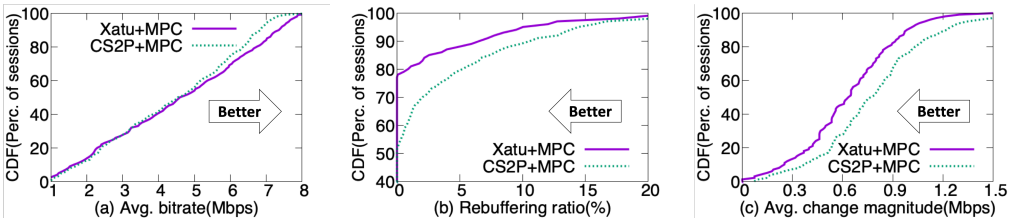


Fig. 9. Individual metrics of QoE-lin for Xatu+MPC and CS2P+MPC.

in the Pensieve experiments, the server queries the Pensieve model to decide the bitrate. The client then requests the appropriate chunk from the video server. Based on the static features (ISP, CDN etc.) of the particular video trace being emulated, the appropriate HMM model is consulted in CS2P experiments, while the static features are used in the model lookup process for the Xatu experiments.

We used one of the reference videos from DASH in all our experiments which has a total length of 192 seconds [5], with chunk durations of 4 seconds and supporting the bitrates {895, 2600, 4729, 9104} Kbps which are corresponding to 480p, 720p, 1080p and 1440p resolution respectively.

The ABR server, video hosting server and a video client run on the same 8-core, 4 Ghz, Intel i7 desktop with 12 GB RAM running Ubuntu 16.04. To emulate different network conditions between servers and client, we used the Chrome DevTools API [8]. This allows us to emulate the network throughput and latency between the client and servers using the Chrome-Remote-Interface based on our traces [2]. All our testbed experiments use a client buffer of 1 minute like previous work [43].

Like prior work[18, 57, 68], we used QoE-lin to summarize video performance of an ABR scheme which is a linear combination of the average bitrate, rebuffering ratio and the average of bitrate change magnitude. Specifically, $\texttt{QoE-lin}\,(P1, P2) = \frac{1}{C} * \left[\Sigma_{t=1}^{C}\,(R_t - P2 * V_t) - P1 * T\right]$ where C is a number of chunk downloaded in a video session, R_t and V_t are a bitrate and a bitrate change magnitude (Mbps) for a chunk t, T is a total rebuffering time (sec) for the video session, and $P1$ and $P2$ are scaling penalties applied to rebuffering and bitrate change magnitude. We use $P1 = 9.1$ and $P2 = 1$ as our default penalties since the maximum bitrate of our video is 9.1 Mbps.

Xatu vs. CS2P: ABR algorithm impact. Following the analysis, we next evaluate whether the improved prediction accuracy of Xatu indeed translates to better performance in terms of video delivery metrics by comparing the performance of the MPC algorithm when integrated with Xatu (Xatu+MPC), and when integrated with CS2P (CS2P+MPC) using the testbed described above. We use a random subset of 500 traces chosen from the testing set of the *Primary DataSet* (§5.2), but only considering traces that have an average chunk download rate less than 10 Mbps since the highest bitrate of the video is 9.1 Mbps similar to previous works [18, 43, 68].

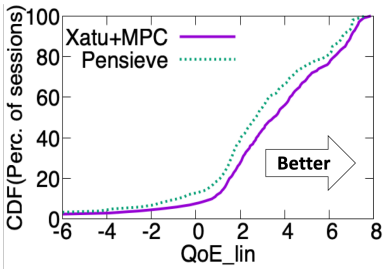


Fig. 10. QoE-lin of Xatu+MPC and Pensieve from the testbed emulation. Note that many features of Xatu are disabled for fair comparison.

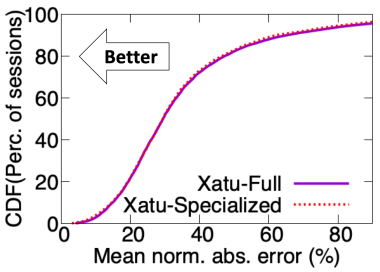


Fig. 11. Ability of Xatu trained on traces with a larger range of chunk download rates to specialize to a smaller range.

Figure 8 shows the CDF of QoE-lin from the emulation testbed on the testing data (§5.2). Xatu+MPC improves the median and 90%ile QoE-lin by 38.9% and 13.2% respectively over CS2P+MPC. Figure 9(a), Figure 9(b) and Figure 9(c) show CDFs of the individual components of QoE-lin, – the average bitrate (Mbps), rebuffering ratio (%) and the average bitrate change magnitude (Mbps) – for the two schemes. While Xatu+MPC maintains similar average bitrate compared to CS2P+MPC, it reduces the number of sessions that have a rebuffering event by 26% and improves the median of the average bitrate change magnitude by 17.4%.

Xatu+MPC vs. Pensieve. We next compare Xatu+MPC to Pensieve, an ABR algorithm that combines reinforcement learning with deep learning to select bitrates for each chunk. Pensieve has been shown to out-perform MPC, when a harmonic mean predictor is used [43]. We compare its performance with MPC when the more sophisticated Xatu predictor is used.

Comparison methodology. We perform the comparison with a subset of 9K traces chosen from the 27K traces in the *Primary DataSet*. Our comparisons use a smaller set because the Pensieve implementation involves CPU-based training and the training time grows with the number of traces. We used the Pensieve implementation provided by the authors [11] and retrained a Pensieve model using the methodology described in [18, 43] for our dataset.

The current implementation of Pensieve does not consider static features (ISP, CDN etc.) and does not consider TTFB. To ensure a fair comparison, and ensure both approaches consider comparable input features, we consider a limited version of Xatu, where we disabled all static features, and only considered chunk size and download time as the temporal features as Pensieve uses.

We trained both Xatu and Pensieve picking 60% of the 9K traces randomly as the training set, and picking 20% for the validation and another 20% for the testing set (with the training, validation and testing sets being the same for both schemes). We only considered traces in the testing set with average chunk download rates under 10 Mbps and select a random subset of 500 of these traces to evaluate the two approaches in the testbed experiment.

Results. Figure 10 shows the CDF of QoE-lin of Xatu+MPC and Pensieve from the testbed video streaming emulation on the testing set. Xatu+MPC improves the median and 90%ile QoE-lin by 29.2% and 5.8% respectively. We hypothesize that a key contributing factor to this result is a recent finding that Pensieve is unable to specialize to different ranges of chunk download rates [18]. Specifically, Akhtar et al. [18] showed that the performance with Pensieve on traces with mean chunk download rates in the range $(0, T_1)$, $T_1 > 0$, degrades when a model trained on chunk download rates in the range $(0, T_2)$ is used relative to a model trained on the range $(0, T_1)$, where $T1 < T2$. A natural question then is how effectively can Xatu specialize?

Specialization with Xatu. To explore this question, we build two models of Xatu: (i) Xatu-Full, which is trained with all traces in the full throughput range from 0 to 100Mbps; and (ii) Xatu-Specialized, trained only with a subset of traces in the training set with average throughput in the range 0 to 10Mbps. Both models used the version of Xatu without static features and with limited temporal features as described above. We then evaluated the prediction accuracy achieved by both Xatu models on the validation set, only considering traces in the range 0 to 10 Mbps. Figure 11 shows the mean NAE achieved with the two models. The curves are almost indistinguishable and the degradation with Xatu-Full is modest. The median of the mean NAE across sessions increased only 0.12% and while the 90%ile increased by 5.2%. We hypothesize that Xatu specializes better that Pensieve, and this is potentially because it is solving an easier problem (prediction of chunk download times) compared to Pensieve which seeks to determine best bitrates to pick. More advances may be needed to get a deeper understanding of the true ability of a neural network model to specialize in sub-populations of the data.

6.1 Xatu vs. Fugu

Fugu is a recent system developed by Yan et al. [66], which comprises two key components:

• **FuguABR.** This refers to a new ABR algorithm developed in [66]. In contrast to the MPC algorithm discussed above, FuguABR uses a stochastically optimal controller which takes in a probabilistic distribution of download times. Further, rather than optimise for the `QoE-lin` metric above, FuguABR considers SSIM, a perceptual measure of picture quality [64].

• **FuguNN.** This refers to a neural network architecture proposed in [66] to predict the download time of video chunks. FuguNN has the following differences relative to Xatu. i) FuguNN predicts a probability distribution of download times rather than a point value; ii) FuguNN only considers temporal features and does not consider static features; (iii) FuguNN does not model TTFB; and (iv) FuguNN uses a fully connected neural network that only considers the last few chunks, unlike Xatu which uses an LSTM that models entire sequences (time series over many chunks).

In this section, we integrate Xatu with FuguABR, and compare the performance with the base Fugu system (FuguABR and FuguNN). The purpose of our comparisons is two fold. First, we seek to evaluate the benefits of Xatu on one of the most recently proposed ABR algorithms, and evaluate Xatu's ability to benefit different ABR algorithms. Second, we seek to evaluate the benefits of neural network architecture used in Xatu with the one used in FuguNN.

Since FuguABR requires a probability distribution of download times, we design XatuDist by adding uncertainty quantification to Xatu. XatuDist modifies the loss function of Xatu to optimise for predictions that are Gaussian distributed rather than a point estimate as before —the mean and standard deviation are predicted by Xatu. In order to more fairly compare XatuDist to FuguABR, we turn off static features in XatuDist since FuguNN does not use them. Moreover, to ensure the temporal features in FuguNN and XatuDist are the same, we only use download time and chunk size in both systems, and do not use TTFB in XatuDist as a feature.

Comparison Methodology. We conducted our evaluations using the Fugu code provided by the authors [7]. Since Fugu focuses on SSIM, we use a modified QoE metric, which we refer to as `QoE-SSIM`. This metric is computed in a similar fashion as `QoE-lin`, with the primary difference that video quality is measured using SSIM rather than just the raw bitrate. We use the default setting of the coefficients for the various terms in the QoE term as used in Fugu [66]. Both FuguNN and XatuDist are trained using the dataset described in Section 5.2. We used the same evaluation setup used by Fugu [66], which involved emulating traces using Mahimahi [49], and a 10 minute

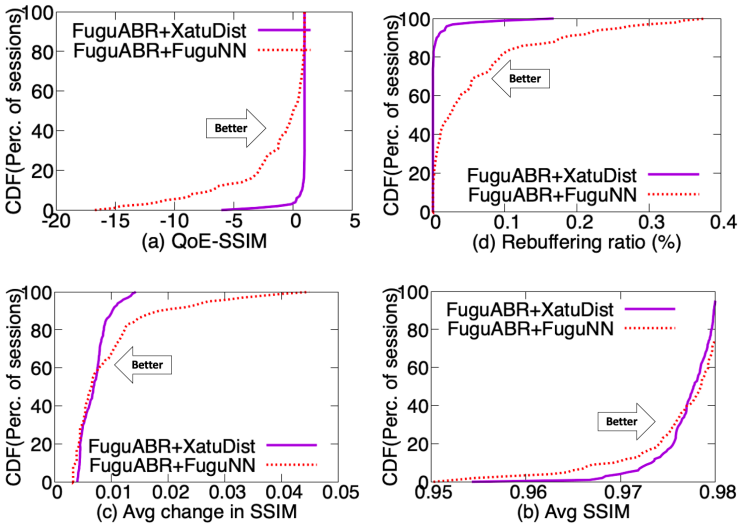


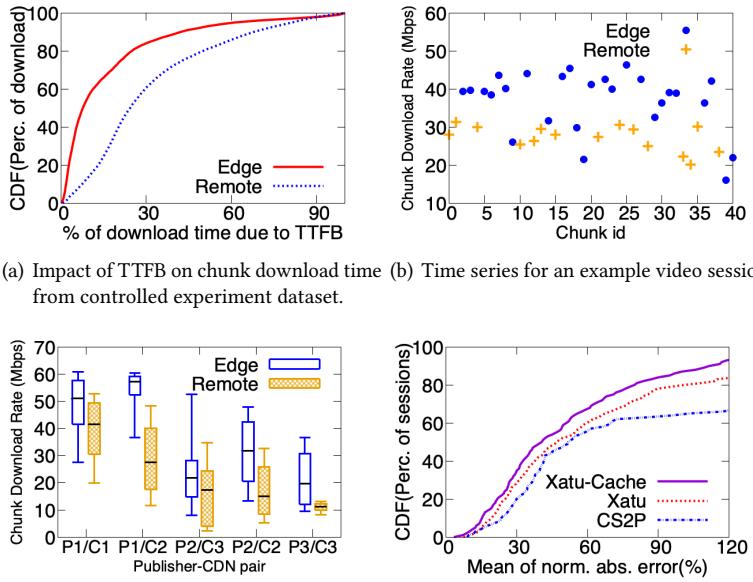
Fig. 12. Benefits of XatuDist over FuguNN when integrated with FuguABR.

pre-recorded video clip. We use traces in the testing set with average chunk download rates that do not significantly exceed the maximum video bit rate (4.1 Mbps) like previous works [18, 43, 68].

Results. Figure 12(a) shows the CDF of QoE_SSIM across sessions for FuguABR + FuguNN and FuguABR + XatuDist. The figure shows that when XatuDist is used, `QoE-SSIM` is greatly improved across all percentiles, with the benefits particularly pronounced over the last 20% of the sessions – for instance, the $20^{th}\%ile$ `QoE-SSIM` with FuguNN is -2.89, while it is 0.92 with XatuDist. Figure 12(b) shows that FuguABR + XatuDist achieves significantly lower rebuffering compared to FuguABR + FuguNN. The median rebuffering ratio for XatuDist is close to 0, while FuguNN has a median rebuffering ratio of about 2%, with the benefits even stronger at higher percentiles. Figure 12(c) shows that the average change in SSIM across chunks is significantly reduced with FuguABR + XatuDist for the last 40% of chunks. Finally, Figure 12(d) show FuguABR + XatuDist achieves similar SSIM as FuguABR + FuguNN. Overall, the results show that (i) the performance of FuguABR is significantly improved when using XatuDist rather than FuguNN; and (ii) validate that Xatu can benefit multiple ABR algorithms. Finally, note that Xatu can potentially achieve further benefits if static features were used.

7 VALIDATION WITH ADDITIONAL DATASET AND EXTENSIBILITY OF XATU TO ADDITIONAL FEATURES

Our results so far have been shown with a dataset of video session data collected from real users (§3). In this section, we further validate these observations using a separate dataset obtained using a controlled measurement study by streaming 500 videos from three popular video publishers that have an Alexa top 100 US rank [16] (Twitch, Vimeo and ESPN). The data is collected over a home network during a 7 day period [31]. One advantage of this dataset is that it not only includes the chunk download time, chunk size, and TTFB, but also includes information regarding which CDN layer the video chunk was served from. Determining the CDN layers involves adding special headers in the HTTP requests sent to each CDN, and interpreting the values of the appropriate headers in the HTTP responses [14, 15, 17, 31].



(a) Impact of TTFB on chunk download time from controlled experiment dataset.

(b) Time series for an example video session.

(c) Distribution of chunk download rate for *Edge* and *Remote* servers.

(d) Xatu's improvement in prediction accuracy with additional information.

Fig. 13. (a) Impact of TTFB on chunk download time for chunks are served from (*Edge* or *Remote*) from controlled experiment dataset, and (b)-(d) benefits of exposing where objects are served from (*Edge* or *Remote*), and ability of Xatu to leverage such information.

Knowledge of the CDN layer helps since the TTFB of a chunk depends on the latency of the client to the CDN server, as well as the time taken by the CDN infrastructure to retrieve a chunk. CDNs are typically organized as a hierarchy of caches [51]. A user request that arrives at a server in an edge cluster could "hit" at that server (referred to as *Edge*), or experience a cache miss in which case the request may be directed to other upstream servers in the CDN hierarchy, and if necessary, ultimately the CDN origin, or origin server (for simplicity, we referred to all layer above *Edge* as *Remote*). Thus, the TTFB depends on whether the object experiences a cache hit in the CDN, and if so, which layer.

Validating TTFB observations. Figure 13(a) considers all chunks across all the sessions, and presents the fraction of the download time of the chunk that may be attributed to TTFB. Two curves are shown, one corresponding to chunks served by an *Edge* server, and another corresponding to chunks served by a *Remote* server. Clearly, TTFB accounts for a large fraction of the download time in this dataset as well, reaffirming the observations in §3. Further, TTFB is a more dominant factor for chunks served from a *Remote* server.

Figure 13(b) shows the time series for an example session corresponding to a popular video with more than 228K views. The X-axis indicates the chunk id, and the Y-axis shows the download rate associated with each chunk. The different symbols indicate whether each chunk was served from an *Edge* or a *Remote* server. The graph shows that even within a session, chunks may be served from either an *Edge* or a *Remote* server with no obvious pattern. Further analysis showed that 73.1% of video sessions corresponding to relatively popular videos with more than 10K views involved video chunks retrieved from both an *Edge* and a *Remote* server. A more in depth analysis is conducted in [31] which further breaks down *Remote* into an upstream CDN layer and origin, and characterizes TTFB by CDN layer. [31] also shows that although a larger fraction of the first

few chunks of a video session are served from the edge, chunks later in the session are also served from different CDN layers. Further, it is not uncommon for consecutive chunks to be served from different locations.

Figure 13(c) shows how the chunk download rate depends on where video chunks are served from across all our traces. For each combination of publisher and CDN (e.g., P1/C1 denotes publisher P1 and CDN C1), we show two boxplots that correspond to the distribution of chunk download rate for chunks being served from an *Edge* or a *Remote* server. Note that each publisher could use multiple CDNs. Across all combinations, the chunk download rate is significantly better when objects are served from an *Edge* server.

Xatu performance. We next evaluate the benefits of Xatu over CS2P on this dataset. We use 400 of the 500 traces as our training set, and the remaining 100 traces as a validation set. Since the dataset was not sufficiently large, our CS2P model did not cluster video sessions. For fairness, we disabled all static features on Xatu. Figure 13(d) shows the mean NAE across sessions in the validation set with both schemes. Xatu continues to show significant benefits over CS2P. These arise because Xatu models the dependence on TTFB, and uses a neural network rather than a HMM.

Extensibility of Xatu. We next show the potential to provide better predictions with Xatu when information not traditionally available to video players can be leveraged. We add an additional feature to Xatu which consists of a single bit that indicates whether the next video chunk will be served from an *Edge* or a *Remote* server (we refer to this version as Xatu-Cache). Like with other schemes, we disabled static features on Xatu-Cache, and present the mean NAE across sessions in Figure 13(d). Xatu-Cache shows a noticeable benefit over Xatu indicating the extensibility of Xatu, and its ability to easily include additional information. In contrast, it is not easy to include such additional information with a HMM-based approach such as CS2P (§3.2).

Architectural changes to facilitate sharing of the information above is an important question in its own right. In the above example, a CDN could provide a video client with information on whether the next chunk can be fetched from an *Edge* server or not. Alternately, if the CDN server were itself to run the ABR algorithm (e.g., [18, 30, 41, 43]), it may be easier to share the information. Nevertheless, the results show the importance of identifying and utilizing underlying information for chunk download rate, and the benefits of an approach like Xatu that can easily leverage such information when available.

8 RELATED WORK

ML in adaptive video streaming. Fugu, a recent system [66] proposes both a new ABR algorithm and a prediction approach using a fully connected neural network, unlike Xatu which uses an LSTM. We have shown Xatu outperforms FuguNN (§6.1). Additionally, Xatu shows how to jointly consider static and temporal features, and highlights the importance of considering TTFB. Several works [23, 25, 26, 61], most notably Pensieve [43], propose ABR algorithms based on Reinforcement Learning (RL) or bayesian bandits [19] to decide which bitrates to select. In contrast, Xatu uses a neural network for the more limited task of predicting chunk download times alone. We have shown that MPC combined with Xatu out-performs Pensieve (§6), potentially because Pensieve does not specialize as well [18]. Understanding the underlying causes, and developing ML models for the more complex task of bitrate selection may benefit from recent developments [44].

Complementary video related research. Recent works [21, 56] have pointed out that smaller chunk sizes may observe poorer throughput than larger ones owing to TCP slow start effects. Ghasemi et al [32] has observed that CDN latency can impact the performance of video streaming sessions but does not discuss how to address the issues. In contrast, we observe that TTFB is an

important factor that contributes to chunk download times, and chunk download rates need not coincide with network throughput. Further, we develop a practical prediction approach based on our observations, and show the benefits on ABR algorithms.

Oboe [18] develops ways to tune ABR algorithm parameters to achieve better performance, and is complementary to Xatu which focuses on prediction. Oboe's offline parameter selection requires iterating over parameter choices for a given ABR and throughput prediction scheme over synthetic traces corresponding to different network states (Guassian traces with different mean and standard deviation). It is not clear how to adapt Oboe's offline parameter learning to work with the maximum likelihood optimization of most data-driven machine learning methods such as CS2P, FuguNN, or Xatu. Combining Oboe with such methods is an interesting avenue for future research. While we have shown Xatu benefits two representative ABR algorithms, using a robust prediction framework like Xatu could be beneficial to recently proposed ABRs [28, 52, 54] as well. Deep neural networks have been used to learn a mapping from a low quality video to a higher quality version to reduce dependency on bandwidth [67]. Prior work [39] has shown the theoretical possibility of oscillations when video is streamed with a caching server. In contrast, we present measurements to show video chunks in a session may be served from different CDN layers, and focus on improving prediction.

Throughput prediction. Prior work has explored TCP network throughput prediction using Support Vector Regression (SVR) [47] and auto-regression [33], or developed approximate analytical models of TCP throughput [50]. Newer congestion control protocols like BBR [22] estimate network throughput. However, these works focus exclusively on TCP network throughput. We have shown many other factors impact chunk download times such as TTFB, and the interplay with cache hierarchy. Further, we have compared our approach with CS2P [57] which has been shown to out-perform many of these techniques. While SVR techniques can include both static and dynamic features as a concatenated joint input, Xatu's approach offers better interpretability (see the discussion around Figures 5(c) and 5(d)). Other work [72] analyzes the benefits of throughput prediction for video streaming [72] but does not consider other factors, or provide a solution.

LSTMs in other domains and recent advances. LSTMs[34] have been widely applied to many domains including language modeling and speech recognition [24, 27, 27, 46, 65]. Recently, LSTMs have been applied to prediction tasks in networking [20, 45, 48, 60, 69]. Beyond applying LSTMs, Xatu introduces a neural network architecture that combines static and temporal features in an interpretable manner, and empirically explores its benefits(§4.1,§5.3). We chose LSTMs over Transformer-based models [27, 62, 70] since transformers use permutation-equivariant representations which are ill-aligned with time series predictions (which require highly permutation-sensitive models).

Recent work [58] empirically shows that adding static features besides dynamic features in LSTMs does not necessarily improve performance in some synthetic tasks. Their approach is limited to concatenating static features after LSTM outputs, and uses random forest as the only classification method. In contrast, Xatu's approach involves an attention mechanism, and concatenation before LSTM inputs in deep neural networks. Zhu et al. [71] also directly concatenate static features with dynamic features extracted by a LSTM, but use regularized logistic regression in a real world task for churn prediction, and shows that static information is helpful. In [35, 40], a LSTM is pre-trained using unsupervised learning, and the LSTM output from dynamic features is directly concatenated with static features for a single classification task. In our context, we have a temporal regression task at every step rather than a single classification task at the last step. Further, we train dynamic feature extractor (e.g., LSTM) and static feature extractor (e.g., fully connected) jointly by attention mechanism. In [29], static and dynamic outputs are directly concatenated before being fed into the last fully connected layer for a temporal classification task. In contrast, we

focus on a temporal regression task that needs a more advanced architecture for point estimates and uncertainty quantification.

9 CONCLUSION

In this paper, we make three contributions. First, we have shown that prediction frameworks for video streaming (i) must not only consider network throughput, but also a richer set of temporal features including TTFB; and (ii) should avoid apriori clustering sessions based on static features. Second, we present Xatu, a general framework to achieve these goals which combines LSTMs with a new gated mask neural network mechanism, to jointly learn a neural network sequence model with an interpretable automatic session clustering method. Third, our evaluations show Xatu achieves better prediction accuracies, and demonstrate Xatu's benefits on multiple ABR algorithms. Relative to CS2P, Xatu improves the median of the mean NAE across sessions by 23.8%. When integrated with MPC, Xatu improves the median `QoE-lin` across sessions by 38.9%, outperforming Pensieve. When integrated with FuguABR, Xatu reduces the 80^{th} percentile rebuffering ratio from 10% to 0%, significantly improving `QoE-SSIM` relative to FuguNN. We show Xatu's benefits on multiple datasets, and its extensibility to new features such as the CDN layer a chunk is served from.

10 ACKNOWLEDGEMENTS

We thank our shepherd, Zhi-Li Zhang, and the anonymous reviewers for their feedback which greatly helped improve the paper. We thank Humberto La Roche for his encouragement of this work. This work was funded in part by the National Science Foundation (NSF) Awards ICE-T:RC 1836889, IIS-1943364 and CCF-1918483, Purdue Integrative Data Science Initiative and Cisco.

REFERENCES

[1] Can I stream Netflix in ultra hd? https://help.netflix.com/en/node/13444.
[2] Chrome Remote Interface. https://github.com/cyrus-and/chrome-remoteinterface.
[3] Cisco Visual Networking Index: Forecast and Trends, 2017–2022 White Paper. https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.html.
[4] Cisco Visual Networking Index: Forecast and Trends, 2017–2022 White Paper. https://apps.fcc.gov/edocs_public/attachmatch/FCC-18-10A1.pdf.
[5] DASH IF Test Assets Database. http://testassets.dashif.org/#testvector/list.
[6] DASH Industry Forum: Dash.js. http://dashif.org/reference/players/javascript/1.4.0/samples/dash-if-reference-player/.
[7] Fugu Github. https://github.com/StanfordSNR/puffer.
[8] Google-Chrome: Chrome DevTools Protocol. https://chromedevtools.github.io/devtools-protocol/tot/Network/.
[9] hmmlearn. https://hmmlearn.readthedocs.io/en/latest/#.
[10] New research reveals buffer rage as tech's newest epidemic. https://www.prnewswire.com/news-releases/new-research-reveals-buffer-rage-as-techs-newest-epidemic-300237001.html.
[11] Pensieve Github. https://github.com/hongzimao/pensieve.
[12] Principal component analysis. https://en.wikipedia.org/wiki/Principal_component_analysis.
[13] PyTorch. https://pytorch.org/.
[14] Reduce CloudFront Latency "X-Cache: Miss from cloudfront". https://aws.amazon.com/premiumsupport/knowledge-center/cloudfront-latency-xcache/.
[15] Understanding cache HIT and MISS headers with shielded services. https://docs.fastly.com/guides/performance-tuning/understanding-cache-hit-and-miss-headers-with-shielded-services.
[16] US Alexa Rank. https://www.alexa.com/topsites/countries/US.
[17] Using akamai pragma headers to investigate or troubleshoot akamai content delivery. https://community.akamai.com/customers/s/article/Using-Akamai-Pragma-headers-to-investigate-or-troubleshoot-Akamai-content-delivery?language=en_US.
[18] Z. Akhtar, Y. S. Nam, R. Govindan, S. Rao, J. Chen, E. Katz-Bassett, B. Ribeiro, J. Zhan, and H. Zhang. Oboe: Auto-tuning video abr algorithms to network conditions. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, pages 44–58, New York, NY, USA, 2018. ACM.

[19] B. Alt, T. Ballard, R. Steinmetz, H. Koeppl, and A. Rizk. Cba: Contextual quality adaptation for adaptive bitrate video streaming (extended version). *arXiv preprint arXiv:1901.05712*, 2019.

[20] A. Azzouni and G. Pujolle. Neutm: A neural network-based framework for traffic matrix prediction in sdn. In *NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium*, pages 1–5. IEEE, 2018.

[21] M. Bartulovic, J. Jiang, S. Balakrishnan, V. Sekar, and B. Siñopoli. 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*, pages 192–198, Palo Alto, CA, USA, 2017. ACM Press.

[22] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson. Bbr: Congestion-based congestion control. *ACM Queue, 14*, pages 20−−53, 2016.

[23] F. Chiariotti, S. D'Aronco, L. Toni, and P. Frossard. Online learning adaptation strategy for dash clients. In *Proceedings of the 7th International Conference on Multimedia Systems*, MMSys '16, pages 8:1–8:12, New York, NY, USA, 2016. ACM.

[24] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

[25] M. Claeys, S. Latré, J. Famaey, T. Wu, W. Van Leekwijck, and F. De Turck. Design of a q-learning-based client quality selection algorithm for http adaptive video streaming. In *Proceedings of the 2013 Workshop on Adaptive and Learning Agents (ALA), Saint Paul (Minn.), USA*, pages 30–37, 2013.

[26] M. Claeys, S. Latré, J. Famaey, T. Wu, W. Van Leekwijck, and F. De Turck. Design and optimisation of a (fa) q-learning-based http adaptive streaming client. *Connection Science*, 26(1):25–43, 2014.

[27] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[28] A. Elgabli, V. Aggarwal, S. Hao, F. Qian, and S. Sen. Lbp: Robust rate adaptation algorithm for svc video streaming. *IEEE/ACM Transactions on Networking*, 26(4):1633–1645, 2018.

[29] C. Esteban, O. Staeck, S. Baier, Y. Yang, and V. Tresp. Predicting clinical events by combining static and dynamic information using recurrent neural networks. In *2016 IEEE International Conference on Healthcare Informatics (ICHI)*, pages 93–101. IEEE, 2016.

[30] A. Ganjam, F. Siddiqui, J. Zhan, X. Liu, I. Stoica, J. Jiang, V. Sekar, and H. Zhang. C3: Internet-scale control plane for video quality optimization. In *12th Symposium on Networked Systems Design and Implementation NSDI '15)*, pages 131–144, 2015.

[31] E. Ghabashneh and S. Rao. Exploring the interplay between cdn caching and video streaming performance. In *2020 IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 2020.

[32] M. Ghasemi, P. Kanuparthy, A. Mansy, T. Benson, and J. Rexford. Performance characterization of a commercial video streaming service. IMC '16, New York, NY, USA, 2016. Association for Computing Machinery.

[33] Q. He, C. Dovrolis, and M. Ammar. On the predictability of large transfer tcp throughput. In *ACM SIGCOMM Computer Communication Review*, volume 35, pages 145–156. ACM, 2005.

[34] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[35] T.-C. Hsu, S.-T. Liou, Y.-P. Wang, Y.-S. Huang, et al. Enhanced recurrent neural network for combining static and dynamic features for credit card default prediction. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1572–1576. IEEE, 2019.

[36] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, pages 187–198, New York, NY, USA, 2014. ACM.

[37] A. Jabri, A. Joulin, and L. Van Der Maaten. Revisiting visual question answering baselines. In *European conference on computer vision*, pages 727–739. Springer, 2016.

[38] J. Jiang, V. Sekar, and H. 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 '12, pages 97–108, New York, NY, USA, 2012. ACM.

[39] D. H. Lee, C. Dovrolis, and A. C. Begen. Caching in http adaptive streaming: Friend or foe? In *Proceedings of Network and Operating System Support on Digital Audio and Video Workshop*, page 31. ACM, 2014.

[40] A. Leontjeva and I. Kuzovkin. Combining static and dynamic features for multivariate sequence classification. In *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 21–30. IEEE, 2016.

[41] X. Liu, F. Dobrian, H. Milner, J. Jiang, V. Sekar, I. Stoica, and H. Zhang. A case for a coordinated internet video control plane. *ACM SIGCOMM Computer Communication Review*, 42(4):359–370, 2012.

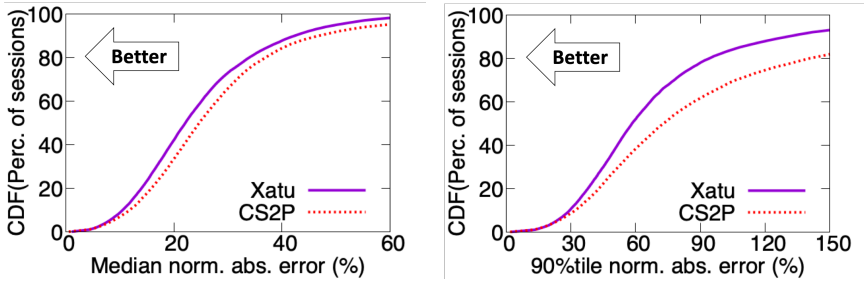
[42] M. Malinowski, M. Rohrbach, and M. Fritz. Ask your neurons: A neural-based approach to answering questions about images. In *Proceedings of the IEEE international conference on computer vision*, pages 1–9, 2015.

[43] H. Mao, R. Netravali, and M. Alizadeh. Neural adaptive video streaming with pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 197–210. ACM, 2017.

[44] H. Mao, S. B. Venkatakrishnan, M. Schwarzkopf, and M. Alizadeh. Variance reduction for reinforcement learning in input-driven environments. In *7th International Conference on Learning Representations ICLR '19*, 2019.
[45] L. Mei, R. Hu, H. Cao, Y. Liu, Z. Han, F. Li, and J. Li. Realtime mobile bandwidth prediction using lstm neural network. In *International Conference on Passive and Active Network Measurement*, pages 34–47. Springer, 2019.
[46] S. Merity, N. S. Keskar, and R. Socher. Regularizing and optimizing lstm language models. *arXiv preprint arXiv:1708.02182*, 2017.
[47] M. Mirza, J. Sommers, P. Barford, and X. Zhu. A machine learning approach to tcp throughput prediction. In *ACM SIGMETRICS Performance Evaluation Review*, volume 35, pages 97–108. ACM, 2007.
[48] A. Narayanan, S. Verma, E. Ramadan, P. Babaie, and Z.-L. Zhang. Making content caching policies' smart'using the deepcache framework. *ACM SIGCOMM Computer Communication Review*, 48(5):64–69, 2019.
[49] R. Netravali, A. Sivaraman, K. Winstein, S. Das, A. Goyal, and H. Balakrishnan. Mahimahi: A lightweight toolkit for reproducible web measurement. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, page 129–130, New York, NY, USA, 2014. Association for Computing Machinery.
[50] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling tcp throughput: A simple model and its empirical validation. *ACM SIGCOMM Computer Communication Review*, 28(4):303–314, 1998.
[51] S. Puzhavakath Narayanan, Y. S. Nam, A. Sivakumar, B. Chandrasekaran, B. Maggs, and S. Rao. Reducing latency through page-aware management of web objects by content delivery networks. In *ACM SIGMETRICS Performance Evaluation Review*, volume 44, pages 89–100. ACM, 2016.
[52] Y. Qin, R. Jin, S. Hao, K. R. Pattipati, F. Qian, S. Sen, C. Yue, and B. Wang. A control theoretic approach to abr video streaming: A fresh look at pid-based rate adaptation. *IEEE Transactions on Mobile Computing*, 2019.
[53] M. Seo, A. Kembhavi, A. Farhadi, and H. Hajishirzi. Bidirectional attention flow for machine comprehension. *ICLR*, 2017.
[54] K. Spiteri, R. Sitaraman, and D. Sparacio. From theory to practice: Improving bitrate adaptation in the dash reference player. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 15(2s):67, 2019.
[55] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman. Bola: Near-optimal bitrate adaptation for online videos. In *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*, pages 1–9. IEEE, 2016.
[56] P. C. Sruthi, S. G. Rao, and B. Ribeiro. Pitfalls of data-driven networking: A case study of latent causal confounders in video streaming. In *ACM Sigcomm workshop on Network Meets AI and ML (NetAI)*, 2020.
[57] Y. Sun, X. Yin, J. Jiang, V. Sekar, F. Lin, N. Wang, T. Liu, and B. Sinopoli. Cs2p: Improving video bitrate selection and adaptation with data-driven throughput prediction. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 272–285, 2016.
[58] M. Tadayon and Y. Iwashita. Comprehensive analysis of time series forecasting using neural networks. *arXiv e-prints*, pages arXiv–2001, 2020.
[59] G. Tian and Y. Liu. Towards agile and smooth video adaptation in dynamic http streaming. In *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies, CoNEXT '12*, 2012.
[60] H. D. Trinh, L. Giupponi, and P. Dini. Mobile traffic prediction from raw data using lstm networks. In *2018 IEEE 29th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, pages 1827–1832. IEEE, 2018.
[61] J. van der Hooft, S. Petrangeli, M. Claeys, J. Famaey, and F. De Turck. A learning-based algorithm for improved bandwidth-awareness of adaptive streaming clients. In *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 131–138. IEEE, 2015.
[62] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is All you Need. In *NIPS*, 2017.
[63] D. Wang and E. Nyberg. A long short-term memory model for answer sentence selection in question answering. In *ACL*, volume 2, pages 707–712, 2015.
[64] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004.
[65] W. Xiong, L. Wu, F. Alleva, J. Droppo, X. Huang, and A. Stolcke. The microsoft 2017 conversational speech recognition system. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5934–5938. IEEE, 2018.
[66] F. Y. Yan, H. Ayers, C. Zhu, S. Fouladi, J. m. Hong, K. Zhang, P. Levis, and K. Winstein. Learning in situ: a randomized experiment in video streaming. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 495–511, 2020.
[67] H. Yeo, Y. Jung, J. Kim, J. Shin, and D. Han. Neural adaptive content-aware internet video delivery. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI) 18)*, pages 645–661, 2018.
[68] X. Yin, A. Jindal, V. Sekar, and B. 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 '15,

London, United Kingdom, 2015.
[69] C. Zhang and P. Patras. Long-term mobile traffic forecasting using deep spatio-temporal neural networks. In *Proceedings of the Eighteenth ACM International Symposium on Mobile Ad Hoc Networking and Computing*, pages 231–240. ACM, 2018.
[70] Y. Zhang, V. Zhong, D. Chen, G. Angeli, and C. D. Manning. Position-aware attention and supervised data improve slot filling. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 35–45, 2017.
[71] F. Zhu, X. Song, C. Zhong, S. Fang, R. Bouchard, V. N. Fontama, P. Singh, J. Gao, and L. Deng. Churn prediction using static and dynamic features, Sept. 6 2018. US Patent App. 15/446,870.
[72] X. K. Zou, J. Erman, V. Gopalakrishnan, E. Halepovic, R. Jana, X. Jin, J. Rexford, and R. K. Sinha. Can accurate predictions improve video streaming in cellular networks? In *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications*, pages 57–62. ACM, 2015.

A APPENDIX



(a) The median NAE (%) for each session is computed, and a CDF is plotted to summarize results across sessions.

(b) The 90^{th} percentile NAE (%) for each session is computed, and a CDF is plotted to summarize results across sessions.

Fig. 14. The median and 90^{th} percentile NAE across sessions for Xatu and CS2P.

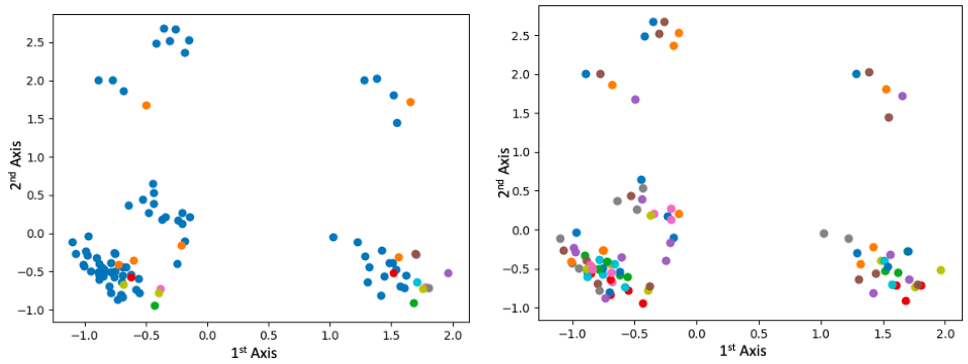
Prediction error metrics (per session)	Improvement (in a median session)
Mean of absolute error	15.5%
Median of absolute error	16.2%
90%tile of absolute error	13.8%
Mean of square error	25.2%
90%tile of square error	10.1%

Table 1. Median improvements of Xatu over CS2P in various prediction error metrics.

Xatu vs. CS2P: Prediction accuracy with various error metrics (§5.3). §5.3 focused on the mean NAE across chunks for each session for Xatu and CS2P. We have also considered the median NAE across chunks, and 90%ile NAE across chunks for each session. Figure 14(a) shows the CDF of the median NAE for both schemes. Figure 14(b) shows the CDF of the 90%ile NAE (%) for the two schemes. The results continue to show the benefits of Xatu, though the results are even more prominent for the 90%ile.

Table 1 summaries improvements in several other prediction error metrics. For instance, we obtained the result in the first row as follows. For each scheme, we computed the mean of the absolute error across chunks for each session, and considered the median across sessions. The

table shows that Xatu achieves an improvement of 15.5% for this metric. The other rows may be interpreted similarly. Regardless of error metrics, Xatu constantly improves prediction accuracy ranging from 10.1% to 25.2% over CS2P.



(a) 2D projection of $z^{(j)}$ for each ISP cluster through PCA.

(b) 2D projection of $z^{(j)}$ for each city cluster through PCA.

Fig. 15. Understanding interpretability in Xatu using PCA

Interpreting static features from Xatu (§5.3). In §5.3, we saw results obtained by using Principal Component Analysis (PCA). Figure 15(a) and Figure 15(b) augment the key ideas presented by Figure 5(c) and Figure 5(d). The dots in Figure 15(a) and Figure 15(b) are colored by city and ISP. The figures do exhibit some interesting patterns in terms of how these features further separate combinations with the same CDN and ToD (e.g., the clusters corresponding to the ISP colored orange are typically to the right of the clusters corresponding to the ISP colored blue within each group). However, overall the results suggest that more prominent variation across models occurs based on the CDN and ToD.

Received August 2021; revised October 2021; accepted November 2021