Popularity Prediction of Facebook Videos for Higher Quality Streaming

Linpeng Tang, ***Princeton University;*** **Qi Huang and Amit Puntambekar,** ***Facebook;***
Ymir Vigfusson, ***Emory University & Reykjavik University;*** **Wyatt Lloyd,** ***University of Southern California & Facebook;*** **Kai Li,** ***Princeton University***

https://www.usenix.org/conference/atc17/technical-sessions/presentation/tang

This paper is included in the Proceedings of the 2017 USENIX Annual Technical Conference (USENIX ATC '17).

July 12–14, 2017 • Santa Clara, CA, USA

ISBN 978-1-931971-38-6

Open access to the Proceedings of the 2017 USENIX Annual Technical Conference is sponsored by USENIX.

Popularity Prediction of Facebook Videos for Higher Quality Streaming

Linpeng Tang*, Qi Huang♭, Amit Puntambekar♭, Ymir Vigfusson†, Wyatt Lloyd‡,♭, Kai Li*

*Princeton University, †Emory University/Reykjavik University, ‡University of Southern California, ♭Facebook Inc.

Abstract

Streaming video algorithms dynamically select between different versions of a video to deliver the highest quality version that can be viewed without buffering over the client's connection. To improve the quality for viewers, the backing video service can generate more and/or better versions, but at a significant computational overhead. Processing all videos uploaded to Facebook in the most intensive way would require a prohibitively large cluster. Facebook's video popularity distribution is highly skewed, however, with analysis on sampled videos showing 1% of them accounting for 83% of the total watch time by users. Thus, if we can predict the future popularity of videos, we can focus the intensive processing on those videos that improve the quality of the most watch time.

To address this challenge, we designed Chess, the first popularity prediction algorithm that is both scalable and accurate. Chess is scalable because, unlike the state-of-the-art approaches, it requires only constant space per video, enabling it to handle Facebook's video workload. Chess is accurate because it delivers superior predictions using a combination of historical access patterns with social signals in a unified online learning framework. We have built a video prediction service, ChessVPS, using our new algorithm that can handle Facebook's workload with only four machines. We find that re-encoding popular videos predicted by ChessVPS enables a higher percentage of total user watch time to benefit from intensive encoding, with less overhead than a recent production heuristic, e.g., 80% of watch time with one-third as much overhead.

1 Introduction

Video is increasingly a central part of people's online experience. On Facebook alone, there are more than 8 billion video views each day [2]. Clients stream these videos by progressively downloading video chunks from a provider according to an adaptive bitrate (ABR) [33, 39] algorithm. ABR algorithms strive to dynamically select the version of a video with the highest bitrate a connection can sustain without pausing. Higher bitrates provide higher quality, but are larger and thus require clients to have higher-bandwidth connections. The different versions of the video used by ABR algorithms are typically generated when a video is uploaded [3]. Generating the different versions for the large volumes of videos uploaded to Facebook each day requires a large fleet of servers.

There is a trade-off between the amount of computation spent processing a video to prepare it for streaming and the quality of experience for viewing that video. Videos uploaded to Facebook are by default encoded to a small number of standard versions with FFmpeg [16]. However, investing in more computation can improve playback experience by improving or increasing the choices for the ABR algorithm. First, more computation can improve the choices by further compressing a video at a fixed quality. For instance, Facebook's QuickFire engine [1] uses up to 20× the computation of the standard encoding to produce a version of the video with similar (or higher) quality that is ~20% smaller than the standard encoding. Second, more computation can increase the choices for the streaming algorithm by generating more versions of the video at different bitrates. In both cases, added computation increases the highest quality version of a video that can be streamed for some users.

Unfortunately, it is infeasible to compute the highest-quality encodings for all videos. Using QuickFire and increasing the number of versions of each video, for example, would require a fleet at several tens the scale of the already large processing fleet at Facebook. Fortunately, video popularity is highly skewed, with 1% of the videos accounting for over 80% of the *watch time*, i.e., the time users spend viewing video. This skew enables us to achieve most of the quality improvement with only a fraction of the computation by generating the highest-quality encodings for only the most popular videos.

The challenge in exploiting this insight is in scalably and accurately predicting the videos that will become popular. State of the art popularity prediction algorithms [9, 10, 45] are accurate but do not scale to handle the Facebook video workload because they keep per-video state that is linear in its past requests. Simple heuristics that exploit information from the social network scale, but are not accurate. For example, predicting popular videos based on owner like count requires 8× more resources to cover 80% of watch time than what would be needed with clairvoyant predictions, which only runs QuickFire encoding on videos with the largest future watch time.

We overcome this challenge with Chess—Constant History, Exponential kernels, and Social Signals—the

first scalable and accurate popularity prediction algorithm. CHESS is scalable because it uses constant per-video state, needing only ~20GB to handle the Facebook video workload. CHESS is accurate: it outperforms even the non-scalable state-of-the-art algorithm. Two key insights led to CHESS. First, we approximate the history of all de-identified past accesses to a video with exponentially-decayed watch time (§4.1) in a few fixed-size time windows, each of which is not highly accurate but small and fast to compute. Second, we combine those constant sized historical features through a continuously updated neural network model to obtain state-of-the-art accuracy, and then further improve it by leveraging social network features—e.g., the like count of video owner—while remaining scalable.

We validate CHESS's scalability by building CHESSVPS, a video prediction service based on CHESS, that requires only four machines to provide popularity prediction for all of Facebook's videos. CHESSVPS has been deployed, providing query-based access to new predictions updated every ten minutes, although its predictions are not yet used to inform encoding choices in production.

Our evaluation compares CHESS against the state-of-the-art non-scalable prediction algorithms, simple scalable heuristics, and a clairvoyant predictor using traces of Facebook's video workload. We find CHESS delivers higher accuracy than all achievable baselines, and provides QuickFire-encoded videos for more user watch time with less re-encoding. Compared to the heuristic currently used in production, CHESS improves the watch time coverage of QuickFire by 8%–30% using same CPU resources for re-encoding. To cover 80% of watch time, CHESS reduces the overhead from 54% to 17%.

The contributions of this paper include:

- The case for video popularity prediction services to improve streaming quality. (§3)
- The design of CHESS, the first scalable and accurate popularity prediction algorithm. (§4)
- The implementation of CHESSVPS, a prediction service for Facebook videos that uses only four machines. (§5)
- An evaluation using Facebook's workload that shows CHESS achieves state-of-the-art prediction accuracy, and delivers high watch time coverage for QuickFire with low CPU overhead from re-encoding. (§6)

2 Background

The workflow of videos on Facebook, which starts with an upload and finishes with streaming, is shown in Figure 1. When a video is uploaded, it is immediately encoded with the H.264 codec to a few standard versions for streaming [40]. The encoded files are durably stored in a backend [5, 32]. In addition, the original upload is kept

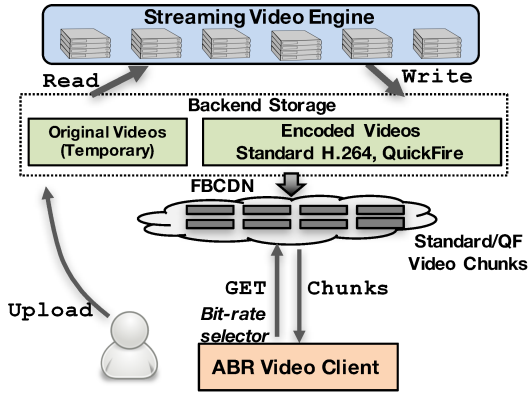


Figure 1: The workflow of videos on Facebook. Additional processing in the Streaming Video Engine can lead to higher quality video delivery to the client by giving the ABR algorithm better choices.

for several days during which it can be re-encoded with QuickFire, used to generate more versions, or both.

Videos are shown to users by a player that downloads progressive chunks of the video from a content distribution network [22, 38]. The player dynamically tries to stream the highest quality version of a video it can without pausing using an ABR algorithm [33, 39]. There are a variety of ABR algorithms [23, 24, 25, 42], but they typically estimate the bandwidth of a user's connection and then select the largest bitrate that is less than that bandwidth.

Generating additional bitrate versions of a video thus improves quality for some users. For example, consider two versions of a video with bitrates of 250 Kbps and 1 Mbps. Generating a third version with a bitrate of 500 Kbps would improve quality for all users with bandwidth between 500 Kbps and 1 Mbps. This is one way additional processing can yield higher-quality video streaming.

Another way to improve video quality is by generating more compressed versions of a video that yield similar or higher video quality at a lower bitrate. FFmpeg's H.264 encoding offers several preset parameters that range from "ultrafast" to "veryslow". Moving to slower encodings yields more compressed versions with the same quality. Facebook's QuickFire [1, 41] technology provides a more extreme trade-off. It intelligently tries many encodings for each chunk of a video, and picks the smallest one with similar or higher quality—client-side decoding is unaffected because each chunk is H.264 compatible. QuickFire can be configured to try 7–20 encodings; we use 20 in this work due to its higher compression.

We quantified this processing/bitrate trade-off for the FFmpeg presets and QuickFire for 1,000 randomly selected videos uploaded to Facebook in one month of 2016. The results of this experiment confirmed that more processing can be used to find better-compressed versions of a video at the same quality. In particular, using QuickFire

takes 20× the processing of "veryslow"[1] but yields a 21% reduction in bitrate for the same quality. This in turn increases the quality of video for some users. For example, consider a 1 Mbps "veryslow" encoding. Generating the QuickFire encoding would yield the same quality at ~800 Kbps. Users with bandwidth between 800 Kbps and 1 Mbps could then stream this higher quality version.

More processing improves the quality of videos that users can stream. Maximally processing all videos would require increasing the already huge number of processing machines by 1-2 orders of magnitude, which is infeasible. Our goal in this paper is to instead extract most of the benefit of using the maximum processing on all videos, but without requiring a substantially larger fleet of machines. We next explain how a scalable and accurate video popularity prediction service helps meet this objective.

3 Motivation and Challenges

This section makes the case for a video popularity prediction service and lays out the challenges of building one, including the need to be quick, accurate, and scalable.

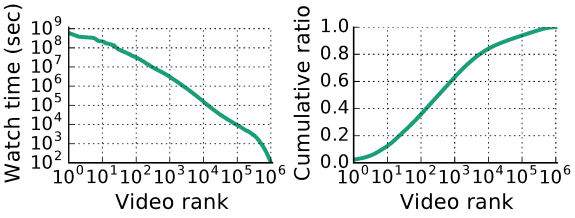
3.1 High Skew Motivates Prediction

Predicting the popularity of videos is compelling because it can guide more processing to where it can do the most good. A small core of videos in Facebook's workload account for most of the time spent watching videos. Thus, if we know what videos will be watched the most in the future, we can focus additional processing on them.

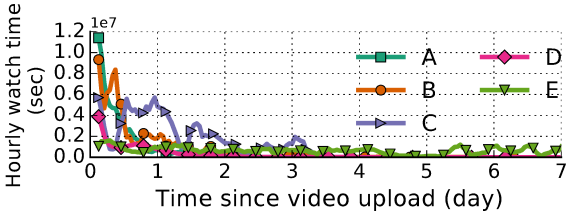
Figure 2a quantifies the skew of Facebook's video workload with the watch time of 1 million randomly sampled videos in one month. The left sub-figure shows the watch time of each unique video, ordered by popularity rank in a log-log scale. For example, the most popular video in the sampled trace has 13 years of watch time in one month, while the $10,000^{\text{th}}$ most popular video out of the million is watched for 42 hours. The shown distribution of watch times follows a power-law with exponent $\alpha = 1.72$. (Related work has shown that access to Facebook photos also follows a power-law distribution with $\alpha = 1.84$ [22].)

The right sub-figure of 2a shows the potential benefit from exploiting this skew. The cumulative ratio of video watch time represented by videos with a given rank or higher is depicted. For example, the top 0.1%/1% of videos account for 62%/83% of the watch time, respectively. Thus, if we use the maximum processing on only 1% of videos we would benefit from increased streaming quality for over 80% of all video watch time. The cumulative watch time ratio is an upper bound on the benefits of popularity prediction because it ranks videos based on their exact accesses, i.e., it represents the benefit from having perfect predictions at the time a video is uploaded.

[1]We could not directly measure the processing time of QuickFire so we approximate it as 20× that of "veryslow" because it encodes each chunk of the video ~20 times



(a) Watch time distribution of sampled videos



(b) Hourly watch time of five example videos

Figure 2: Facebook videos access patterns.

3.2 Prediction is Challenging

The difficulty in exploiting the skew lies in being able to quickly, accurately, and scalably predict the popularity of individual videos. Prediction needs to be quick so not many views of the video are missed while waiting for prediction results. Prediction needs to be accurate so computation is spent on videos that reap the most benefit. Finally, prediction needs to be scalable so it can handle video workloads at a global scale like Facebook.

To motivate each of these points, we manually examined the access pattern of 25 videos in the one month trace with rank 10,000–10,024, i.e., they are near the cut-off for the top 1% of popularity and all have a similar total watch time. Figure 2b shows the access patterns of 5 representative videos. The other 20 videos have access patterns that resemble one of the depicted patterns.

The Need for Quick Prediction The video access pattern peaks quickly for videos A – D in Figure 2b. This indicates we need our video prediction service to run quickly. If our prediction takes longer than the interval between when a video is uploaded and when it peaks, then much of the watch time will have already taken place when the prediction is ready. To further demonstrate this point, we analyzed the full one month trace and found that the most popular 1/4/16 hours of each video accounts for 6.3%/19%/29% of watch time. Previous work on video popularity [18, 36] considered popularity on a daily basis. Such methods, if applied on our workload, would have a large delay in prediction and would miss a significant portion of the total watch time. Instead, we aim for quick predictions on the order of minutes.

The Need for Accurate Prediction The variety of access patterns in Figure 2b suggests that accurately predicting future watch time will be challenging. Prediction needs to be accurate so additional computation is used where it will be the most useful. Using simple heuristics based on features from the social network is quick, but unfortunately is not accurate. For instance, a recent production heuristic was to re-encode a video if the like count of the owner exceeded 10,000. As our evaluation in Section 6 shows, this heuristic is inaccurate: it requires re-encoding 8× as many videos as a clairvoyant predictor to cover 80% of the video watch time. Our goal is to provide predictions with higher accuracy so higher watch time coverage can be achieved with fewer resources.

The Need for Scalable Prediction Video popularity prediction for Facebook must be scalable because there are tens of millions of videos uploaded each day. Identifying popular videos thus requires predicting the popularity of a large active set of videos. In the video prediction service described in Section 5 we track 80 million videos. The previous state of the art in popularity prediction, SEISMIC, is accurate but unfortunately does not scale to our workload because it stores the timestamp and watch time of each past request. This linear per-video state would require ~10TB of memory to make predictions for 80 million videos, and methods requiring more features per request [10] have an even larger memory usage.

4 The Chess Prediction Algorithm

Achieving high watch time coverage through additional processing requires quick, accurate, and scalable prediction of video popularity. This section describes the core of Chess, the novel prediction algorithm we designed with these goals in mind. We focus on three key features:

1. Harnessing past access patterns with constant space and time overhead.
2. Combining different features in a unified model.
3. Efficient online training using the recent access data.

4.1 Utilizing Past Access Patterns with EDWT

A common theme in popularity prediction is exploiting past access patterns [13, 36, 43, 45]. The state of the art approaches do so by modeling behavior as a self-exciting process that predicts future accesses based on all past accesses. A past access at time t is assumed to provide some *influence* on future popularity at time τ, as modeled by a kernel function $\phi(\tau - t)$. The kernel function, ϕ, is a probability density function defined on $[0, +\infty)$, and it is commonly chosen to be a decreasing function, so that a session's influence is initially high and gradually converges to zero over time.

Self-exciting processes predict future popularity—i.e., watch time—based on the sum of the influence of all past requests from the current time to infinity. Let i be an index over the past viewing sessions of a video. Let t_i and x_i be the corresponding timestamp and watch time, respectively of the session. Then, for the purposes of ranking different videos, the total future watch time for i is modeled as

$$\tilde{F}(t) = \sum_{t_i \le t} \int_{\tau}^{+\infty} x_i \phi(\tau - t_i) d\tau.$$

One key insight in Chess is using a kernel that allows for efficient updates to popularity predictions. Previous popularity prediction algorithms used power-law kernels that provide high accuracy predictions, but require each new prediction to compute over all past accesses [13, 45]. This requires storage and computation linear in the past requests to each video, which is not feasible in our setting. In contrast, we set ϕ to be the exponential kernel, or $\phi(t) = \frac{1}{w} \exp(-t/w)$, where w represents a time window modeling how long past requests' influence lasts into the future. Such a kernel allows us to simplify the computation of a new prediction to only require the last prediction, $\tilde{F}$, and its timestamp, u, which drastically reduces the space and time overhead. Below is the simplified update rule for a new session with watch time x beginning at time t with a previous session having occurred at time $u < t$. The resulting prediction is the *exponentially decayed watch time* (EDWT):

$$\begin{aligned}
\tilde{F}(t) &= \sum_{t_i \le t} x_i \int_{t}^{\infty} \phi(\tau - t_i) d\tau \\
&= \frac{x}{w} + \sum_{t_i \le u} x_i \exp\left(\frac{-(t - t_i)}{w}\right) \\
&= \frac{x}{w} + \exp\left(\frac{-(t-u)}{w}\right) \sum_{t_i \le u} x_i \exp\left(\frac{-(u - t_i)}{w}\right) \\
&= \frac{x}{w} + \exp\left(\frac{-(t-u)}{w}\right) \tilde{F}(u). \qquad (1)
\end{aligned}$$

4.2 Combining Efficient Features in a Framework

While EDWTs are efficiently computable, they are weaker predictors of popularity than self-exciting processes with more complex kernels as shown in our evaluation (§6). We overcome this limitation of EDWTs with the second key insight in the Chess design: combining many weak, but readily computable, signals through a learning framework achieves high accuracy while remaining efficient. We use a neural network as our learning framework with two types of features as input: stateless and stateful.

Stateless features are quantities that do not change dramatically during the life-cycle of a video. A prediction service does not need to keep any state associated with these features or their past values. Instead it can query

them from the social network at prediction time. For our purposes, the most important are the social features, including the number of likes and friends of the video owner. They also include the video's length, its age, and several other easily queryable social features.

Stateful features are quantities that can vary dynamically throughout the life-cycle of the video. Past access patterns are one type of stateful feature. The changing pattern of the number of comments, likes, shares, saves for later viewing, etc. are all stateful features as well. They are stateful in that a prediction service needs to keep state associated with them between predictions. We use exponential kernels to keep this state constant per-video and we combine four kernels with different time windows—1, 4, 16, and 64 hours—to capture more complex patterns.

We use the stateless and stateful features as input to a 2-layer neural network (NN) with 100 hidden nodes for predicting total future watch time. We find that neural networks reduce the prediction error by 40% compared to linear models, but more complex models, i.e., adding more layers or using more hidden nodes do not further improve accuracy. We initially selected all features from the social network that we thought could provide some signal and then trimmed those that did not have an effect on prediction accuracy. We made features stateless or stateful based on our intuition, e.g., friends of the video owner is stateless because it changes little during the lifetime of the video. We also tried several different sets of time windows for stateful features and settled on 1, 4, 16, and 64 hours as providing the highest accuracy. We did this feature engineering using a setup similar to the single prediction experiments in our evaluation, on a separate and earlier month-long trace.

Another important technique for boosting accuracy is logarithmic scaling of both the feature values and prediction targets. Because these values can vary from 10-10^8 depending on video popularity, they need to be properly scaled to avoid optimization difficulties. Although linear scaling, in the form of standardization [6], is the commonly used method in statistical learning, we find that *logarithmic scaling*, i.e., $x \rightarrow \log(x+1)$, delivers much better performance for our workload. It ensures the model is not biased towards only predicting extremely popular videos, achieves good prediction accuracy across the whole popularity spectrum, and improves the coverage ratio of QuickFire by as much as 6% over linear scaling. We use this method in all our evaluations.

4.3 Efficient Online Model Update

Naively training our model would require a large set of training examples with their full future watch time, which is unknown. To address this issue, we use an *example queue* to generate training examples from the recent past, and use them as approximations for the future. When a video is accessed, its current state is appended to the queue. While the video is in the queue we track its watch time and feature values. Later, when an example is evicted from the queue it becomes training data with the difference in watch time between its entry and eviction used as the target future watch time. As an added benefit, because examples keep entering and being evicted from the queue, the prediction model is continuously updated at a constant learning rate to keep up with changes in the workload.

The example queue needs to be carefully designed in order to minimize the memory and CPU overhead while achieving the best model accuracy. We found that two design parameters are key to balancing this trade-off: prediction horizon and example distance. Section 6.3 investigates the effect of varying each parameter and shows that setting them properly leads to high accuracy with low memory and CPU overhead.

The *prediction horizon* is the time difference between entry and eviction of examples from the queue. In other words, an example is evicted and becomes training data when its age in the queue exceeds the prediction horizon. A larger horizon provides a better approximation of total future watch time, but it also results in a longer queue with higher memory usage. For our workload, a prediction horizon of 6 days achieves a good tradeoff with high accuracy and low overhead.

We found our example queue was flooded by data points from the most popular videos due to the skewed power law distribution in video access. Many of these data points were effectively redundant and did not help improve accuracy. This is because the input values and the prediction target will be very similar for the same video at two nearby time points. We skip these redundant examples using an admission policy that only allows a new example into the queue if the difference between its timestamp and the most recent example for the same video is greater than a threshold. We call this threshold the *example distance* D because it ensures there is at least D time between all examples of the same video. Although this alters the training data distribution, we find $D = 2\text{h}$ achieves high accuracy while greatly reducing memory overhead, due to the high skew and large volume of data.

5 The Implementation of ChessVPS

To make video popularity predictions continuously available we implemented the Chess video prediction service (ChessVPS). ChessVPS validates the scalability of our design by providing popularity prediction for Facebook's video workload while running on only four machines.

Figure 3 provides a high-level view of the architecture of ChessVPS. The service uses 8 workers distributed across 4 machines to generate predictions on the full workload. The key steps in the process are: 1) ingesting access logs,

2) querying for additional features, 3) making predictions, 4) serving predictions, and 5) updating the models.

Ingesting Access Logs Video accesses on Facebook are logged to Scribe [15]. We ingest the access logs by continuously streaming them from Scribe. To handle this streaming load—as well as distribute prediction computation—we use 8 worker processes on 4 machines. The access logs in Scribe are sharded based on video ID and each worker streams one-eighth of the shards.

Querying for Additional Features Each worker augments the access logs with the additional features it queries from TAO [7], Facebook's cache for the social graph. Current values of these features are already stored in TAO, e.g., the number of likes of a video is stored in TAO so it can be presented along with the video. Stateless features are directly added to each access of a video. The 4 exponentially decayed counters for each stateful features, with varying time windows, are updated upon every access, and the values added to the feature set as well.

We reduce the overhead from querying for additional features in three ways. First, we batch queries and only dispatch them once we have ingested 1000 accesses. Second, we deduplicate queries for the same video in a batch. Third, we cache results from TAO for 10 minutes, which reduces the load we impose on TAO by over 50%.

Making and Serving Predictions Each worker maintains a table with its most recent predictions for the top 10 million most popular videos in its shard. The 80 million videos in all shards encompass the actively accessed video working set on Facebook. After the worker queries TAO for additional features, it updates the exponentially decayed kernels, and feeds all feature values into the neural network to calculate a prediction—the design of Chess has enabled us to do all this in real-time, on a small set of machines. This prediction is then used to update the video's entry in the table. Every 10 minutes each worker scans its table and sorts videos based on their predictions. An aggregator in each machine collects the top 1 million videos from the collocated workers, and then it broadcasts its predictions to all aggregators and waits for their predictions. Once an aggregator has the top 1 million predictions from all 8 workers, it merges and sorts them. It then caches the aggregated predictions and uses them to answer requests for the next 10 minutes. Other services, e.g., a re-encoding service, can query any worker to learn the videos that we predict to be the most popular.

Updating the Model and Memory Overhead To reduce space overhead we maintain one model and example queue per machine (shared between two workers). We use an example queue with a prediction horizon of 6 days and an example distance of 2 hours to keep the memory overhead low. We further reduce the memory overhead of the example queue by only admitting a consistently sampled 30% of the videos to it—this proportionally reduces the queue size, without causing any model overfitting.[2] The resulting example queue consumes ~6 GB of memory per machine, or ~24 GB in total.

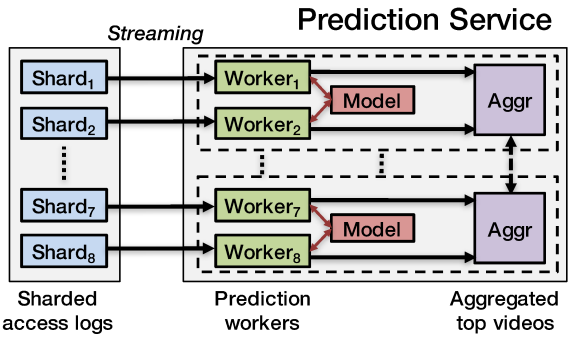


Figure 3: Chess video prediction service architecture.

Each video has 12 stateless features and 7 stateful features. These features, associated metadata, and current popularity prediction add up to a storage overhead of ~250 bytes per video. Thus, all 80 million videos use ~20GB RAM in total to maintain. This results in a total memory overhead of ~44GB RAM from models and metadata, or only ~11GB RAM per machine. In contrast, if SEISMIC were used instead, the timestamp and watch time of each past request would need to be stored to make predictions, translating to 1.2MB per video on average and ~10TB total memory usage.

6 Evaluation

Our evaluation seeks to answer three key questions for Facebook's video workload:

1. How does the prediction accuracy of Chess compare to the heuristic used in production, the state of the art, and a clairvoyant predictor?
2. What are the effects of our design decisions, such as prediction target scaling, prediction horizon, and example distance, on accuracy and resource usage?
3. How would adopting ChessVPS for production processing decisions impact resource consumption and watch time coverage?

6.1 Experimental Setup

Predictors Table 1 shows the predictors we compare in this evaluation in three groups: baselines, increasing subsets of Chess, and a clairvoyant predictor. Among the baselines, we modified SEISMIC and Initial(1d) to suit our application scenario better, and tuned their parameters

[2] This sampling turned out to be unnecessary, as even without it the memory footprint per machine is still only 25GB.

Predictor	Ranking of videos based on:
Initial(1d) [36]	Watch time in the initial day after upload, or total watch time if less than a day old.
SEISMIC-CF [45]	State of the art popularity prediction using a power-law kernel, with followers of each viewer set to constant for our application.
Owner-Likes	Like count of the video owner. This was recently used in production.
EDWT(4h)	Exponentially decayed watch time with a four hour time window.
NN(EDWT)	Neural network model using only EDWT features with time windows 1h, 4h, 16h, 64h.
CHESS	Neural network model with stateless features (e.g., owner likes) and stateful features (e.g., video views, video likes) made efficient using EDWTs.
Clairvoyant	Total future watch time of each video. This is unattainable in practice.

Table 1: Popularity predictors evaluated on Facebook's video workload in our evaluation.

to yield the best performance on our dataset. The original SEISMIC algorithm needs the number of followers of each retweeter for predicting tweet popularity, which is unsuitable for video watch time prediction on Facebook because a viewer might not share the video after watching and directly influence its followers. Based on a parameter sweep, we settled on a constant 1000 for this setting on our workload, with the ensuing method called SEISMIC-CF—as shown below, its performance remains competitive even with this modification. Initial(1d) [36] originally uses the number of requests—watch time in our case—of the entire first day for predicting popularity, but for our application, if the video is less than 1 day old we use its total watch time to generate a prediction instead of waiting.

Comparing to baselines that represent the state of the art—Initial(1d) and SEISMIC-CF—and a recent production heuristic—Owner-Likes—enables us to quantify how much CHESS improves on the state of the art and would improve production. Comparing increasing subsets of CHESS—EDWT(4h) and NN(EDWT)—allows us to quantify the improvement from each addition to CHESS. Comparing to a clairvoyant predictor allows us to quantify how far CHESS is from a perfect predictor.

Experimental Methods and Workloads We use three experimental methods with progressively more realistic results and time-consuming experiments: single prediction, simulation, and real-time sampled processing. The single prediction method resembles that used by prior work on popularity prediction [18, 45] and enables comparisons with SEISMIC. The simulation method enables us to run many experiments in a reasonable time frame and we validate its results using real-time sampled processing.

Workloads. Single prediction and simulation experiments each use the same 35-day trace of video access as their workload. The trace is comprised of full access logs for a random sample of 1% of videos during 5 weeks. The workload for the real-time sampled processing experiments was the full Facebook video workload.

Single prediction. The memory and computational overhead of SEISMIC[3] made it infeasible for us to run the more realistic simulation (or real-time sampled processing) experiments with it, so we designed the single prediction method to enable evaluation against it. In this method each predictor takes as input the historical information for a video up to a time point and then issues predictions. The predictions are then evaluated using the watch time of the video in the 15 days immediately following the time point.

The input historical information and future watch time of the videos are extracted from the trace as follows. First, we select only the videos in the trace that are accessed on one day at the midpoint of the trace. This limits the size of the prediction to make the experiments feasible. Second, we randomly pick a time point on that day for each video to control for diurnal effects. Finally, we extract the trace up to the time point for each video and the future watch time in the 15 days following the time point.

Simulation. Our main evaluation method is simulation of a video prediction service that runs hourly using our 35-day trace. In each simulation, we replay the whole trace, train our prediction model continuously, and the predictor ranks videos for re-encoding every hour. Once a video is selected for re-encoding, it is recorded in a hash table. The hash table is then queried for each request to see whether the requested video has already been re-encoded before. We use the initial 23 days of the trace to populate the hash table, and report results on the last 12 days.

Real-time sampled processing. Our final evaluation method is the most realistic and follows the description in Section 5. The whole service operates on 4 machines, each with 20 2.8GHz cores and 32GB memory, and processes access logs of all Facebook videos in real time. We then write a client using results from CHESSVPS to make encoding decisions in 10 minute intervals. The whole system was run for a week for warm up and we present the results from the next day.

[3]The implementation of SEISMIC is ~200× slower than CHESS's implementation. However, part of this slowdown stems from SEISMIC being implemented in the R language [45].

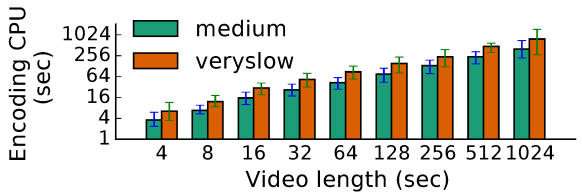


Figure 4: The linear relationship between video length and encoding CPU makes video length a reasonable proxy for encoding CPU.

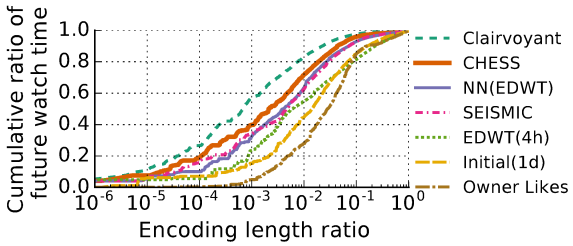


Figure 5: Single prediction results shown by the cumulative ratio of future watch time of the videos selected by each predictor for a given encoding length ratio.

Metrics Our ideal metrics for evaluating predictors would include the future watch time ratio of re-encoded videos and the encoding overhead from doing additional processing on them. Neither of these metrics is feasible for us to collect, but we can gather reasonable approximations of them nevertheless. In the prediction experiments, total future watch time is impossible to collect because there is always more future. Instead we track watch time within a 15-day period because popularity of Facebook videos typically stabilizes in one week (from Figure 2b). In simulations and real-time processing, we keep track of the watch time coverage of re-encoded videos in every hour, and find the coverage ratio stabilizes within 5 days after enough recently popular videos have been re-encoded, so in simulations we have a 23 day warm-up period and report the average coverage ratio in the next 12 days trace, while in real-time processing we wait 1 week before reporting results in the next day. Doing additional processing on all videos is not feasible because it would require the use of a fleet of machines much larger than the current processing fleet. Instead we approximate processing overhead using video length and by doing sampled processing.

Video length is a reasonable proxy for processing CPU. We use video length as our overhead metric for single prediction and simulation experiments because it is fast to compute and a reasonable proxy for processing CPU usage. To demonstrate it is a reasonable proxy we randomly sampled 3000 videos uploaded to Facebook, bucket them by $\log_2$ of their lengths, and show the 20^{th} percentile, median, and 80^{th} percentile CPU usage for FFmpeg "medium" and "veryslow" encodings in each bucket. While there is a large variance in each bucket, the CPU usage is approximately linear in the video length. Statistically this is a strong linear relationship with $R^2 = 0.981$ between length and median CPU usage across the buckets. Based on this observation, in both single prediction and simulation experiments, we rank the videos with each method, and re-encode the top videos until the total length exceeds a threshold (representing a fixed CPU budget). We then compute its ratio to total length of all videos, terming the quantity "encoding length ratio".

Sampled processing. We use measured CPU usage from processing a sample of videos as our overhead metric for the real-time sampled processing experiment. For this experiment, a 0.5% random sample of the selected videos for each predictor ($\approx$ 3000 in total) are re-encoded using QuickFire. At the same time, 5000 videos are sampled from the video uploads that day and encoded with FFmpeg "veryslow". We then calculate the overhead for encoding the selected videos using the measured encoding time for these two sets: let U denote the average FFmpeg "veryslow" encoding time of the sampled video uploads, and Q the average QuickFire encoding time of the videos selected by one method in the sample set, with 95% confidence interval $[Q^-, Q^+]$ (computed using `scikits-bootstrap` [14]). If N is the total number of videos selected by that method, and M the daily video uploads to Facebook, then we estimate the CPU overhead to be $\frac{QN}{UM}$, with confidence interval $\left[\frac{Q^-N}{UM}, \frac{Q^+N}{UM}\right]$, which helps us estimate the variance from sampled processing.

6.2 Single Prediction Experiments

The results of the single prediction experiment that enable us to compare to SEISMIC are shown in Figure 5. The results generally follow the intuition that predictors with more information available to them will make better predictions. For instance, Initial(1d) and Owner-Likes each perform poorly because they use only a single scalar value as their prediction. We highlight two results.

NN(EDWT) is competitive with SEISMIC-CF. EDWT(4h) is a self-exciting process prediction method inspired by SEISMIC with the primary difference being the use of an exponentially decayed kernel that makes it much more resource efficient. The gain in resource efficiency, however, comes with a consistently lower coverage ratio for EDWT(4h) than for SEISMIC-CF. For instance, to achieve 80% coverage EDWT(4h) needs to select 2.9× more minutes of video than SEISMIC-CF.

NN(EDWT) is a combination of four EDWTs in a neural network model. It performs slightly worse (up to 6% lower watch time coverage) than SEISMIC-CF when encoding a very small fraction of videos ($< 0.1\%$). When encoding a more typical fraction of videos ($> 0.1\%$), however,

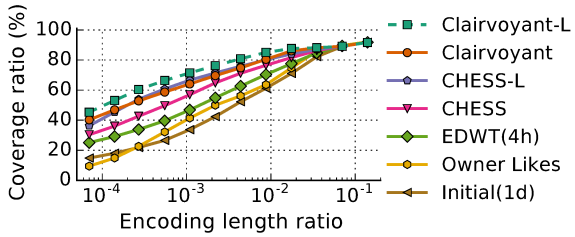


Figure 6: Simulation results shown by the watch time coverage ratio of videos selected by each predictor for a given encoding length ratio. Clairvoyant-L/Chess-L denotes the corresponding algorithm with scores normalized by video length.

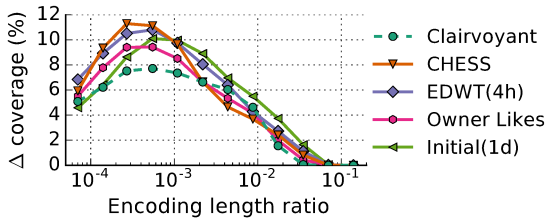


Figure 7: Improvement of coverage ratio through score normalization by video length.

it achieves similar or slightly higher performance than SEISMIC-CF. Both of these methods are based solely on past access patterns, which indicates our learning framework is able to deliver comparable results to a handcrafted algorithm even when only using features of lesser quality and consuming fewer resources.

Chess provides higher accuracy. The full Chess provides the highest watch time coverage of all achievable predictors we evaluated and is the closest to the clairvoyant predictor. Its improvement over SEISMIC-CF is significant: it achieves 40% watch time coverage with 2.0× fewer minutes of video, 60% coverage with 1.8× fewer minutes, and 80% coverage with 1.6× fewer minutes.

6.3 Simulation Experiments

We used simulation experiments to provide a more realistic comparison to other predictors and to investigate the effects of three design parameters: prediction target scaling, prediction horizon, and example distance.

Chess provides higher accuracy. Figure 6 show the watch time coverage of all predictors except SEISMIC-CF which is excluded because of its high memory usage and slow speed. The relative performance of different methods are similar to the single prediction experiment (Figure 5), with Chess and Chess-L outperforming other practical methods, which validates those results. For instance, to reach 80% coverage, Chess-L encodes 2× as many video-minutes as Clairvoyant-L, while Owner-Likes encodes 8×. The overall performance at lower encoding length ratios ($<10^{-3}$), however, improves for two reasons: (1) due to the power-law distribution of popularity, the simulation will include a larger number of the most popular videos than the single prediction experiments that use a smaller sample, (2) in simulations a video is likely re-encoded shortly after gaining popularity, therefore covering more watch time, whereas in the single prediction experiment a random time point is picked to divide the past and future. The second reason also explains why Owner-Likes now outperforms Initial(1d) under many settings even though it did worse in the single prediction experiments. With Owner-Likes, videos are re-encoded at upload time, and so the benefits of re-encoding start accumulating immediately. In contrast, Initial(1d) always waits up to 1 day until a video is popular to select it and misses many of its early views. For most settings, the benefit from higher accuracy in Initial(1d) does not make up for the early views it misses relative to Owner-Likes.

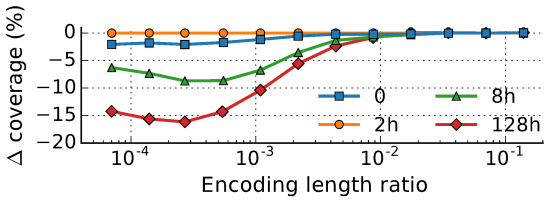
The coverage ratio of different results saturates and converges to the same value when encoding length ratio is above 7%, but because QuickFire takes 20× CPU. This translates to 140% additional CPU usage, a big increase to the already large fleet.

Score normalization by length improves accuracy. Figure 7 shows the increase in coverage of each method with and without score normalization by video length. We find this technique consistently improves the performance of all methods, with Chess seeing the biggest improvement and Clairvoyant seeing the smallest, which reduces the gap between the two.

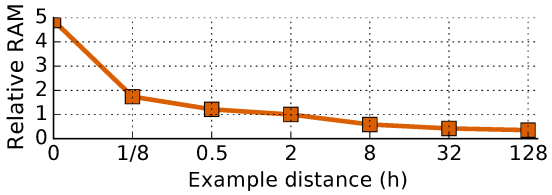
Clairvoyant-L and Chess-L in Figure 6 show the two corresponding methods with scores normalized by video length. Clairvoyant-L achieves the highest coverage ratio consistent with Section 3.1: 80%+ with 1% encoding length ratio, and 70%+ with 0.1% encoding length ratio. Chess-L delivers the best result among all the practical methods, only 6%–8% lower than Clairvoyant-L.

Chess-L improves the coverage ratio of the production baseline, Owner-Likes by 8%-30% with the same encoding length ratio from 0.01%-2%. To achieve 80% coverage ratio, Chess-L only needs to encode 0.9% of total video length, while Owner-Likes needs 2.5%. The results are especially favorable at the middle to lower end of the encoding length ratio. We hope this result motivates the design of new encoding algorithms that utilizes even higher CPU usage to achieves even better compression ratios. Even if this encoding method uses 100× the CPU of FFmpeg, with Chess-L, 64% of the watch time can still be served with only ~10% CPU overhead from re-encoding 0.1% of videos.

Increasing the prediction horizon has diminishing returns and higher memory usage. Due to space limi-



(a) Example distance vs. relative coverage difference.



(b) Memory usage with different example distances.

Figure 8: Effects of example distance.

tations we only summarize the results from varying the prediction horizon. We experiment with horizons of 1h, 1d, 2d, 4d, 6d, 8d, 12d, and `inf`. The coverage is lowest when prediction horizon is as short as 1 hour. It then improves as the prediction horizon increases until 6d, then however, when the horizon is 8d and `inf`, the coverage drops by 1%-2%. Because the training target of our model is the watch time within the prediction horizon, a longer horizon means a better approximation for total future watch time and improves the result. However, when the horizon is too long, the training examples evicted from the queue were created a long time ago, and the stale training data hurts the prediction accuracy.

Meanwhile, the memory usage of the queue grows roughly linearly with the prediction horizon because examples within the horizon are held in the queue. As it provides the highest coverage with modest memory usage in ChessVPS, we choose 6d to be the default prediction horizon in our evaluations.

A short example distance increases coverage and decreases memory usage. Example distance, the minimum time distance between two examples of the same video, is another knob controlling the trade-off between coverage and system resource usage. We have run experiments with values 0, $\frac{1}{8}$h, $\frac{1}{2}$ h, 2h, . . . , 128h, and show the relative coverage compared to the default 2h in Figure 8a, and the simulator's relative memory usage in Figure 8b. We have omitted some lines in the former for clarity but describe the results below.

The memory usage of the queue drops monotonically as the example distance D increases. When D = 2h, we reduce memory by 5× compared to D = 0 (not using the heuristic) because most examples from the popular videos never enter the queue. Interestingly, the coverage ratio increases a little at the same time because the examples

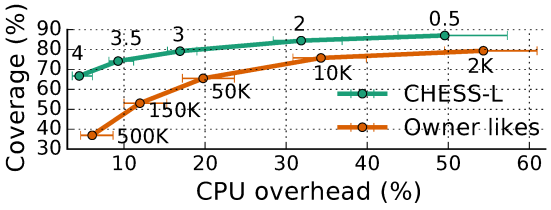


Figure 9: Projected impact of Chess-L compared to Owner-Likes. Encoding score thresholds are annotated for each data point.

skipped are all "duplicates" of the most popular videos; removing them has little effects on training set diversity while making the model less biased towards those videos. This improves the overall performance similar to the effects of *logarithmic scaling*. If D further increases, memory usage continues to drop, though at the expense of the much lower coverage ratio, up to ~15% at 128h. Under such a setting, most examples from even the moderately popular videos are filtered out, and the model fails to deliver accurate predictions. Based on these results we have picked 2h as our default example distance.

6.4 Real-time Sampled Processing

We validate our algorithm and implementation by deploying ChessVPS and running it in real-time with the production access logs. Although the real-world encoding logic is complex and our service is not used by the production encoding pipeline yet, we have implemented a "pseudo client" that queries the service every 10 minutes and issues encoding decisions based on the prediction scores. This way we can monitor the coverage and encoding statistics in real time, and verify its projected impact more realistically. In simulations we ranked the videos every hour and encoded them until the total video lengths reach a threshold, but to more closely resemble the production heuristic here, which re-encodes videos whose owners have more than 10K likes, we also issue re-encoding decisions for videos with prediction scores exceeding a threshold. In the following discussion Chess(α) means Chess with score threshold α, and similarly for Owner-Likes(β).

Figure 9 shows the real-time sampled processing results. Chess-L(3) achieves ~80% coverage ratio as Owner-Likes(10K), while reducing the re-encoding CPU overhead from 54% to 17%. At slightly lower CPU usage, Chess-L(2) improves the coverage of Owner-Likes(10K) from 75.7% to 84.4%. The improvement of Chess-L is greater at lower CPU overhead settings. For example, Owner-Likes(500K) delivers 37% coverage with 6% CPU overhead, whereas Chess-L(4) achieves 66.7% coverage with 4.5% CPU overhead. This is favorable for limited computing budgets, or if we want to apply even more

computing intensive encoding methods or have more encoded versions. The relative performance between the two methods concords with the simulation results shown in Figure 6; the minute differences stem from a changing workload and the logic for different encoding thresholds.

7 Related Work

Our work explores building a scalable and accurate popular video prediction service, with applications on re-encoding for improving streaming quality. In this section we discuss related work on popularity prediction, video quality of experience (QoE) optimization, and caching, which we draw inspiration from for this study.

Popularity Prediction In recent years, the popularity prediction of online content has attracted intense research attention. Simple heuristics like counting requests in the first few hours/days [36], or followers of the owner are fast but inaccurate. Meanwhile, various methods have been proposed for modeling Twitter/Facebook resharing [10, 9, 44]. They usually maximize accuracy, rely on more features and are memory/computation intensive, e.g., requiring to store and scan multiple features of each retweet/sharing when making every prediction. Our method is designed for both accuracy and efficiency, and delivers accurate, real-time prediction for all Facebook videos with a small hardware footprint.

Self-exciting processes have been used for modeling earthquakes [20], YouTube video accesses [13], and Twitter resharing [45]. These methods use variants of power-law kernels and thus store and process all past requests. Instead, we use an exponential kernel to cut per-video memory/computation overhead to $O(1)$. Exponentially decayed metrics are used in other contexts [12, 21]; our contribution is using them for self-exciting processes and appling them to popularity prediction. Furthermore, we are the first to combine multiple exponentially-decayed kernels in a learning framework, which allows us to match the accuracy of a power-law kernel while remaining resource efficient, thus obtaining the best of both worlds.

Video QoE Optimization As videos gain increasing importance in people's online activities, research on improving video streaming QoE has flourished. Many of them focus on the delivery path, e.g., selecting the best bit-rate per chunk in ABR for efficiency, stability and fairness [23, 24, 25, 42], and building a control plane for video delivery [17, 28, 30]. On the upload and encoding path, video codecs have evolved towards using higher computation in exchange for higher compression, from MPEG-2 [19] to the now widely adopted H.264 [34], and gradually moving to the next generation codecs such as VP9 [31] and H.265 [35]. In addition, QuickFire [1, 41] and Netflix per-title encoding [4] try to improve compression of existing codecs by finding the best encoding configuration based on video content as well as resolution. We explore another dimension in video encoding based on feedback from delivery. By applying more processing to popular videos, we optimize the overall trade-off between encoding CPU and streaming QoE.

Caching We find the video re-encoding problem also bears some interesting similarities to caching. By locating hot data in a small but fast storage, caching saves access latency and bandwidth [37]. Meanwhile, by spending more CPU on the popular videos, re-encoding improves the video streaming quality at given network conditions.

Many caching algorithms have been designed to exploit different characteristics of request patterns, including recency (LRU [26]), frequency (LFU [29]), or both (SLRU [27], MQ [46]). The exponentially decayed kernel used as a building block in Chess combines both recency and frequency, and the trade-off is tuned through the time window parameter. Similar to length normalization, size-aware caching [8, 11] also favors smaller items so more can be cached in limited space, improving object hit-ratio.

8 Conclusion

Facebook serves billions of videos views every day and new videos are uploaded at a rapid rate. With limited CPU resources, it is challenging to identify which of these videos would most benefit from re-encoding with computing intensive methods like QuickFire that enhance the viewing experience.

We have described an efficient video popularity prediction service that has the Chess algorithm at its core. Chess achieves scalability by summarizing past access patterns with a constant number of values, and it achieves efficiency by combining the past access patterns and other features in a continuously updated neural network model. Our evaluation show that compared to a recent production heuristic, Chess reduces encoding CPU required by 3× to cover 80% of user watch time with QuickFire.

While the focus of this paper has been popularity prediction for the Facebook video workload, we conjecture that our ChessVPS approach would generalize to efficiently predict popularity in other settings.

Acknowledgments We are grateful to our shepherd Vishakha Gupta-Cledat, the anonymous reviewers of the ATC program committee, Siddhartha Sen, Haoyu Zhang, Theano Stavrinos, and Aqib Nisar for their extensive comments that substantially improved this work. We are also grateful to Sergiy Bilobrov, Minchuan Chen, Maksim Khadkevich, and other Facebook engineers for their discussion on this problem. Our work is supported by Facebook, NSF CAREER award #1553579, and a Princeton University fellowship.

References

[1] QuickFire technology explained @Scale. https://www.facebook.com/atscaleevents/videos/1682906415315789.

[2] Facebook Community Update. https://www.facebook.com/photo.php?fbid=10102457977071041.

[3] Facebook's Streaming Video Engine @Scale Talk. https://www.facebook.com/atscaleevents/videos/1741710496102047/.

[4] A. Aaron, Z. Li, M. Manohara, J. De Cock, and D. Ronca. Per-Title Encode Optimization. http://techblog.netflix.com/2015/12/per-title-encode-optimization.html.

[5] D. Beaver, S. Kumar, H. C. Li, J. Sobel, P. Vajgel, et al. Finding a needle in haystack: Facebook's photo storage. In *USENIX OSDI*, 2010.

[6] C. M. Bishop. Pattern recognition. *Machine Learning*, 128, 2006.

[7] N. Bronson, Z. Amsden, G. Cabrera, P. Chakka, P. Dimov, H. Ding, J. Ferris, A. Giardullo, S. Kulkarni, H. C. Li, et al. Tao: Facebook's distributed data store for the social graph. In *USENIX ATC*, 2013.

[8] P. Cao and S. Irani. Cost-aware www proxy caching algorithms. In *USITS*, 1997.

[9] G. H. Chen, S. Nikolov, and D. Shah. A latent source model for nonparametric time series classification. In *ACM NIPS*, 2013.

[10] J. Cheng, L. Adamic, P. A. Dow, J. M. Kleinberg, and J. Leskovec. Can cascades be predicted? In *ACM WWW*, 2014.

[11] L. Cherkasova. *Improving WWW proxies performance with greedy-dual-size-frequency caching policy*. Hewlett-Packard Laboratories, 1998.

[12] G. Cormode, F. Korn, and S. Tirthapura. Exponentially decayed aggregates on data streams. In *IEEE ICDE*, 2008.

[13] R. Crane and D. Sornette. Robust dynamic classes revealed by measuring the response function of a social system. *PNAS*, 2008.

[14] C. Evans. scikits-bootstrap. https://github.com/cgevans/scikits-bootstrap.

[15] Facebook. Facebok Scribe. https://github.com/facebook/scribe/wiki.

[16] FFmpeg. The FFmpeg project. http://ffmpeg.org.

[17] 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 *USENIX NSDI*, 2015.

[18] G. Gürsun, M. Crovella, and I. Matta. Describing and forecasting video access patterns. In *INFOCOM, 2011 Proceedings IEEE*, pages 16–20. IEEE, 2011.

[19] B. G. Haskell, A. Puri, and A. N. Netravali. *Digital Video: An Introduction to MPEG-2*. Springer Science & Business Media, 1997.

[20] A. G. Hawkes. Spectra of some self-exciting and mutually exciting point processes. *Biometrika*, 1971.

[21] J.-W. Hsieh, T.-W. Kuo, and L.-P. Chang. Efficient identification of hot data for flash memory storage systems. *ACM TOS*, 2006.

[22] Q. Huang, K. Birman, R. van Renesse, W. Lloyd, S. Kumar, and H. C. Li. An analysis of facebook photo caching. In *ACM SOSP*, 2013.

[23] 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. *ACM SIGCOMM Computer Communication Review*, 2015.

[24] J. Jiang, V. Sekar, and H. Zhang. Improving fairness, efficiency, and stability in HTTP-based adaptive video streaming with festive. In *ACM CoNEXT*, 2012.

[25] J. Jiang, V. Sekar, H. Milner, D. Shepherd, I. Stoica, and H. Zhang. CFA: A practical prediction system for video QoE optimization. In *USENIX NSDI*, 2016.

[26] T. Johnson and D. Shasha. 2Q: A Low Overhead High Performance Buffer Management Replacement Algorithm. 1994.

[27] R. Karedla, J. S. Love, and B. G. Wherry. Caching strategies to improve disk system performance. *IEEE Computer*, 1994.

[28] X. Liu, F. Dobrian, H. Milner, J. Jiang, V. Sekar, I. Stoica, and H. Zhang. A case for a coordinated internet video control plane. In *ACM SIGCOMM*, 2012.

[29] S. Maffeis. Cache management algorithms for flexible filesystems. *ACM SIGMETRICS Performance Evaluation Review*, 1993.

[30] M. K. Mukerjee, D. Naylor, J. Jiang, D. Han, S. Seshan, and H. Zhang. Practical, real-time centralized control for cdn-based live video delivery. *ACM SIGCOMM Computer Communication Review*, 2015.

[31] D. Mukherjee, J. Bankoski, A. Grange, J. Han, J. Koleszar, P. Wilkins, Y. Xu, and R. Bultje. The latest open-source video codec VP9-an overview and preliminary results. In *IEEE PCS*, 2013.

[32] S. Muralidhar, W. Lloyd, S. Roy, C. Hill, E. Lin, W. Liu, S. Pan, S. Shankar, V. Sivakumar, L. Tang, et al. F4: Facebook's warm blob storage system. In *USENIX OSDI*, 2014.

[33] I. Sodagar. The MPEG-DASH standard for multimedia streaming over the internet. *IEEE MultiMedia*, 2011.

[34] G. J. Sullivan, P. N. Topiwala, and A. Luthra. The h. 264/avc advanced video coding standard: Overview and introduction to the fidelity range extensions. In *SPIE Optics + Photonics*, 2004.

[35] G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand. Overview of the high efficiency video coding (HEVC) standard. *IEEE CSVT*, 2012.

[36] G. Szabo and B. A. Huberman. Predicting the popularity of online content. *CACM*, 2010.

[37] A. S. Tanenbaum and A. S. Woodhull. *Operating systems: design and implementation*, volume 2. Prentice-Hall Englewood Cliffs, NJ, 1987.

[38] L. Tang, Q. Huang, W. Lloyd, S. Kumar, and K. Li. RIPQ: advanced photo caching on flash for facebook. In *USENIX FAST*, 2015.

[39] T. C. Thang, Q.-D. Ho, J. W. Kang, and A. T. Pham. Adaptive streaming of audiovisual content using MPEG DASH. *IEEE Transactions on Consumer Electronics*, 2012.

[40] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra. Overview of the h. 264/avc video coding standard. *IEEE Transactions on circuits and systems for video technology*, 2003.

[41] WIRED. Facebook acquires QuickFire Networks. https://www.facebook.com/wired/posts/10152676478868721.

[42] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli. A control-theoretic approach for dynamic adaptive video streaming over HTTP. *ACM SIGCOMM Computer Communication Review*, 2015.

[43] T. Zaman, E. B. Fox, E. T. Bradlow, et al. A Bayesian approach for predicting the popularity of tweets. *The Annals of Applied Statistics*, 2014.

[44] T. R. Zaman, R. Herbrich, J. Van Gael, and D. Stern. Predicting information spreading in Twitter. In *ACM NIPS Workshop on computational social science and the wisdom of crowds*, 2010.

[45] Q. Zhao, M. A. Erdogdu, H. Y. He, A. Rajaraman, and J. Leskovec. SEISMIC: A self-exciting point process model for predicting tweet popularity. In *ACM SIGKDD*, 2015.

[46] Y. Zhou, J. Philbin, and K. Li. The multi-queue replacement algorithm for second level buffer caches. In *USENIX ATC*, 2001.