Streaming 360-Degree Videos Using Super-Resolution

Mallesham Dasari*, Arani Bhattacharya†, Santiago Vargas*, Pranjal Sahu*, Aruna Balasubramanian*, Samir R. Das*
*Stony Brook University, †KTH Royal Institute of Technology
*{mdasari, savargas, psahu, arunab, samir}@cs.stonybrook.edu †aranib@kth.se

***Abstract*—360° videos provide an immersive experience to users, but require considerably more bandwidth to stream compared to regular videos. State-of-the-art 360° video streaming systems use viewport prediction to reduce bandwidth requirement, that involves predicting which part of the video the user will view and only fetching that content. However, viewport prediction is error prone resulting in poor user Quality of Experience (QoE). We design PARSEC, a 360° video streaming system that reduces bandwidth requirement while improving video quality. PARSEC trades off bandwidth for additional client-side computation to achieve its goals. PARSEC uses an approach based on super-resolution, where the video is significantly compressed at the server and the client runs a deep learning model to enhance the video to a much higher quality. PARSEC addresses a set of challenges associated with using super-resolution for 360° video streaming: large deep learning models, slow inference rate, and variance in the quality of the enhanced videos. To this end, PARSEC trains small micro-models over shorter video segments, and then combines traditional video encoding with super-resolution techniques to overcome the challenges. We evaluate PARSEC on a real WiFi network, over a broadband network trace released by FCC, and over a 4G/LTE network trace. PARSEC significantly outperforms the state-of-art 360° video streaming systems while reducing the bandwidth requirement.**

***Index Terms*—360° Video, ABR Streaming, Super-resolution.**

I. Introduction

360° video streaming brings an immersive experience by projecting the panoramic content on a virtual display. Its popularity on commercial streaming platforms is on the rise. The key challenge with 360° videos is that they require 8× more content to be downloaded than regular videos for the same perceived quality due to their panoramic nature [11]. Recent work has shown that the bandwidth requirement can be reduced by viewport[1] adaptive streaming where the content to be downloaded is restricted to the user's predicted viewport [32], [41], [42]. A 360° video is divided into *segments* temporally to enable streaming, and each segment is divided spatially into video *tiles* to enable viewport adaptation.

Unfortunately, accurate viewport prediction is difficult; state-of-the-art viewport-adaptive systems only have an accuracy of $\approx 58-80\%$ even for predicting just 1 sec in advance and progressively lower for longer durations [15], [32]. To counter imperfect prediction, additional content beyond the predicted viewport needs to be fetched to avoid missing portions of the viewport at the time of viewing [32] (tile miss). This only addresses the issue partially as this additional content consumes network bandwidth as well.

We describe a different approach for streaming 360° videos for better adaptation: trading off network bandwidth for client-side compute capacity. We design PARSEC (PAnoRamic StrEaming with neural Coding) – a system that fetches low resolution 360° video content over the network and reconstructs the high resolution content at the client by utilizing recent advances in *deep neural networks (DNN) based super-resolution* [14], [24]. While the general idea is promising, there are several challenges in this approach, all of which stem from the large size of the 360° videos. First, much of the 360°content is consumed on mobile devices. Even though current generation mobile devices have improved compute capacity, running a DNN model to reconstruct a high quality 360° video from low resolution input is very slow [22]. Second, the DNN models are large and require considerable network bandwidth [45], which defeats the original motivation. Finally, because the model has to generalize over the entire video, there is a large variance in the quality of the reconstructed videos. Related works that use similar neural techniques [18], [45] do not face any of these problems because they are designed for regular videos that are considerably smaller.

PARSEC exploits two ideas well-suited for 360° videos. First, PARSEC trains super-resolution DNN models over small segments of the video (§III). Use of these small **micro-models** results in three benefits: i) much faster inference rate, ii) addressing viewport prediction inaccuracy by dynamically generating any tile based on user's current viewport, iii) the model transfer over the network is now efficient and can be streamed for each segment, unlike streaming a single large model for the entire video in the beginning. Second, PARSEC pools both compute *and* network resources using a **neural-aware adaptive bitrate (ABR) algorithm** (§IV). For a subset of tiles that spatially partition a given video segment, PARSEC uses the DNN model to locally generate high resolution tiles. For the remaining subset of tiles, PARSEC streams the tiles at high resolution from the server subject to the available network bandwidth. The ABR algorithm determines which tiles to generate locally and which to fetch from the server, given the predicted viewport, available bandwidth and available compute resources. PARSEC formulates this problem as an Integer

†Work done when the author was at Stony Brook University.

[1]The viewport is the portion of the 360° scene that is currently visible to the user.

Linear Program and solves it using a greedy algorithm. Finally, PARSEC combines the neural ABR technique with viewport prediction, and reschedules tile generation by updating the predictions dynamically.

We implement PARSEC on top of GPAC [21], a multimedia library that provides APIs for video coding, rendering and tile packaging in DASH format. We develop the DNN models in Python using Keras [4] and Tensorflow [10] frameworks. We evaluate PARSEC using a 360° video dataset [25] that has traces of 50 user's head movements as they watch 10 videos. We compare the performance with three alternative approaches – Flare [32] and Fan *et al* [15] that are designed for 360° video, and an adaptation of NAS [45] for 360° video.[2]

PARSEC outperforms all three alternatives across all experiments. In terms of video quality of experience (QoE), PARSEC outperforms the state-of-the-art 360° video streaming system, Flare [32], by 37–48% over publicly available broadband network and 4G/LTE traces. PARSEC also improves QoE by 17–28% for experiments over real WiFi networks. When the network is really poor (1 Mbps), the relative performance of PARSEC is even better, outperforming alternatives by 1.8×. Finally, we highlight that PARSEC uses 43% less bandwidth compared to Flare when the network is not the bottleneck, which is more promising when multiple users access the same network (§V).

II. Background, Related Work and Overview

A. 360° Video Streaming

A 360° video is a spherical video where multiple camera directions are recorded simultaneously. The recorded content is projected on a planar surface [1], [2], [49] and then traditional video encoding techniques (e.g., HEVC [48]) and protocols such as DASH [36] are used for streaming. We assume an equirectangular projection [1] as typically used in related work [15], [30], [32], [43].

ABR streaming: For streaming, the video is first segmented across time and then each segment is encoded in a number of different bitrates/quality levels and stored at the server. The different quality levels that are available for each video segment are captured in a manifest file sent from the server to the client. The client (or, sometimes the server) runs an adaptive bitrate (ABR) algorithm [45] to determine the bitrate for streaming based on the available network bandwidth.

Viewport-adaptive 360° video streaming: 360° videos require 8× more content to be downloaded relative to regular video for the same quality, thus significantly increasing bandwidth requirement [11]. To combat this, recent studies use viewport prediction [15], [30], [32], [43]. Each segment of the 360° video, after projection on a 2D plane, is partitioned spatially into *tiles*. Only the tiles in the user's predicted viewport are streamed to the client. A viewport-adaptive streaming system specifies which tiles to fetch according to the predicted viewport, and then uses an ABR algorithm to determine the bitrates to be used for these tiles. Recent studies such as [15], [30], [32], [43], follow a similar architecture.

[2]NAS leverages super-resolution techniques for regular video streaming. We adapt NAS to 360° video streaming by integrating with our viewport prediction.

B. PARSEC's Motivation

PARSEC overcomes the limitation of poor accuracy of viewport prediction, described below, by exploiting underutilized compute capacity on the client device.

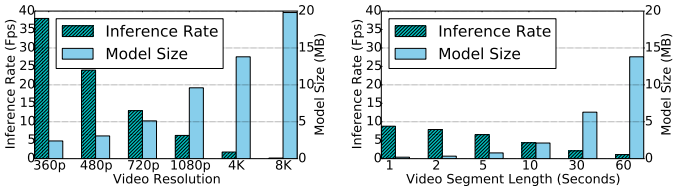
Limitation – Poor accuracy of viewport prediction: Clearly, a perfect viewport prediction can make 360° video as efficient as regular videos in terms of bandwidth usage for the same quality of experience. However, in reality such predictions have poor accuracy. This is hardly surprising as predicting the viewport is same as predicting the user's future visual attention in a panoramic scene – a challenging problem in computer vision [20], [29]. For example, state-of-the-art viewport prediction [15], [32] has an accuracy of $\approx 58-80\%$ for predicting the user viewport just one second in advance. Predictions further in advance exhibit worse accuracy. To accommodate for the possibility of 'tile misses' due to poor viewport prediction some adaptive 360° streaming systems [17], [32] fetch additional tiles (e.g., tiles in the neighborhood of the predicted viewport) at lower resolutions. These non-viewport tiles now compete for bandwidth along with viewport tiles. This presents a tradeoff between the resolution of the content the user actually views and possibility of tile misses (which will incur stalls). This impacts quality of experience. Our evaluation (§V) demonstrates this problem. While viewport prediction is important, the currently available computer vision techniques are still insufficient.

Opportunity – Underutilized compute capacity on client: Mobile devices today have multi-core high-speed CPUs and also a variety of co-processors, specifically fairly capable GPUs that can accelerate a variety of computations amenable to the SIMD paradigm. It is also widely speculated that NPUs (neural processing unit) will be soon available on mobile platforms [9].

Thus, one promising direction is to leverage the underutilized compute capacity to enhance the quality of the video content using neural encoding [18], [45]. Here, the video is down-scaled and compressed significantly to a very low-resolution, and then an appropriately trained DNN model *reconstructs* the high-resolution version of the video. This shifts part of the burden from the network to the processors on the client device. We explore different recent deep learning techniques such as Generative Adversarial Networks (GANs) [16], Autoencoders [33] and Super-resolution [24], and find super-resolution to be the most suitable method in the context of streaming videos in terms of inference and model complexity (see §III).

C. PARSEC Overview and Goal

The goal for PARSEC is to improve the QoE of 360° videos under constrained bandwidth and imperfect viewport prediction. The key idea is to 1) exploit unused compute capacity in the client devices to reconstruct video content



(a) (b)

Fig. 1: Challenges of super-resolution in terms of inference rate and model size. a) Impact of resolution – trained for one minute video, b) Impact of video length – trained on 4K video.

and 2) arrive at an optimal balance of what content to download and what to reconstruct. The reconstruction part uses a *deep learning-based super-resolution approach that is able to reconstruct the content from a down-scaled version*. We show that by exploiting client-side co-processors such as GPU, PARSEC can significantly reduce the pressure on network bandwidth, overcome the poor accuracy of viewport prediction, and improve QoE. There are two key innovations.

Use of micro-models (§III): A straightforward application of super-resolution as used in regular videos [18], [45] does not work well in the 360° case due to very large neural network model size and slowdown in the inference rate (no. of frames that can be reconstructed per sec). In PARSEC, we explore use of *micro-models* (very small models) that i) model small video segment lengths at a time and ii) reconstruct one tile at time for the segment and not the entire panoramic scene. This improves download time and also inference rate. This per-tile approach also allows the flexibility of upscaling from a very low-resolution input (we call this ultra-low resolution or ULR tiles) still keeping the model size reasonable. In our evaluations, we have achieved $64\times$ upscaling (§III-C).

Neural-aware ABR algorithm (§IV): Even with improved inference rate with micro-models, the rate of generating tiles locally is not enough to support high quality video streaming. Thus, PARSEC leverages both network and compute resources to stream 360° videos. PARSEC chooses which tiles to fetch from the server, and which tiles to generate locally. PARSEC's neural-aware ABR algorithm takes into account the variance in tile quality of locally generated tiles, as well as the network and compute capacities. PARSEC incorporates its scheduling algorithm into the viewport adaptive ABR framework. Finally, PARSEC is able to dynamically reschedule tile generation at the client device in response to a change in user's viewport. Because PARSEC can compress tiles to ultra low resolution (ULR), for each video segment, PARSEC downloads all ULR tiles. A combination of the above two makes tile miss relatively uncommon (validated in §V).

III. Streaming 360° Videos using Super-resolution

Super-resolution creates or restores a high resolution image from one or more low resolution images of the same scene [24], [31]. It has been used in surveillance [50], medical imaging [35], [39], and recently in video streaming [45]. Note here that video applications already encode/compress videos to reduce bandwidth demand using traditional techniques. Super-resolution takes this general idea significantly further by providing a trained DNN model that is used by the client to 'infer' the original high-resolution content from a very low-resolution input [24]. The inference exploits the GPU power available at the client end.

A. 360° versus Regular Videos

While super-resolution is indeed promising, one needs to take careful design decisions to make it work effectively. The key challenge here is the large spatial content for 360° videos. Because of the low accuracy of viewport prediction, the entire 360° video needs to be trained so that any tile can be reconstructed on the client depending on the viewport of the user. The result is that the models trained to reconstruct 360° videos are 1) large, creating additional burden to transfer from the server to the client, and 2) have slower inference rate risking real time requirements of playback speed. Attempts to keep model complexity low results in loss of video quality.

To illustrate this, we run a series of micro-benchmarks using the experimental setup described in §V. We train a DNN model similar to NAS [45], a recent video streaming study that uses super-resolution for regular video and trains a single model for the entire video. Video length of 1 minute is used for training, with a target PSNR of 30dB.[3] Figure 1a shows the impact of video resolution on model size and inference rate on the Galaxy S10 phone. The key takeaways from the figure is that: 1) the inference rate for high resolution content such as 4K and 8K videos is prohibitively slow (less than 2 fps), 2) the model size for these high resolutions is very high. Figure 1b complements this analysis by showing how training for different video segment durations impacts the performance metrics. The inference rate is progressively poorer for longer video segments and becomes less than 2 fps for 1 min long video. Even for the smallest video length (1 sec) it falls short (< 10 fps). Also, the model size increases super-linearly with video lengths beyond a certain point (5 sec).

Clearly, a plain use of the super-resolution technique that works well for regular videos is not effective for 360° videos. However, 360° videos provide unique opportunities for optimization. First, the super-resolution can be employed individually for tiles as opposed to the entire panoramic scene and only the tiles *likely* to be in the viewport could be generated thus saving on computational burden of inference.[4] Also, not all tiles in the viewport need to be generated; some can be downloaded using a traditional adaptive streaming technique. We will explore the latter aspect in §IV.

B. Micro-Models for Super-resolution

The insight in PARSEC is to train the model for very short video lengths. We call them micro-models. In the previous

[3]We target at least 30dB because it is the minimum PSNR needed to perceive a good video playback experience [40]

[4]Note that tiling regular videos is not efficient because it introduces unnecessary cross-tile compression overheads [32].

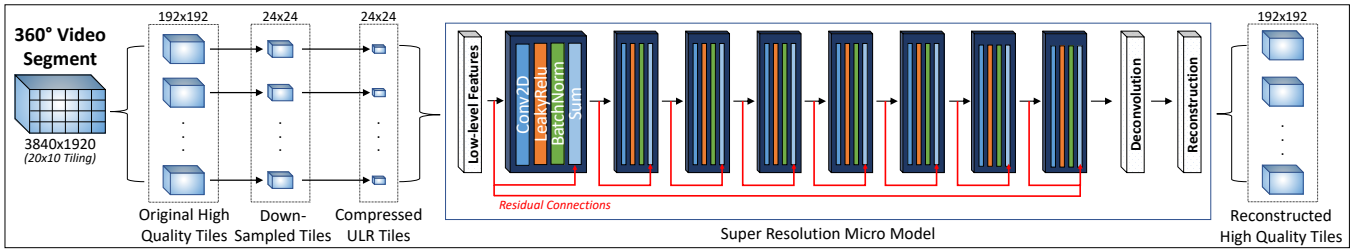
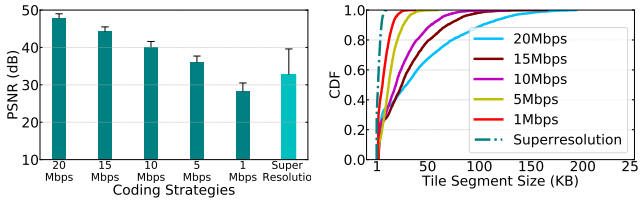


Fig. 2: Super-resolution micro-model architecture (similar to [45]). The model training and ULR tile processing is done offline and transferred to the video server later. The figure shows an example with a tile size of 192×192 down-scaled to 24×24.



(a) (b)

Fig. 3: HEVC video encoding quality and size for different rates vs. the super-resolution approach. The bitrate and quality are computed for each tile with a tile size of 192x192 for one second segment.

subsection we have showed that training the model for shorter video lengths is efficient both in terms of inference rate and model size. In addition, the micro-models are trained at a segment level so that individual tiles can be upscaled dynamically depending on the current viewport of the user. This is especially critical for 360° videos where a user can only see a part of the scene, but this part cannot be accurately predicted in advance. The trade-off is that the client downloads a large number of small (micro) models, but each micro-model is only trained for a short video segment. Downloading small models at regular intervals is more efficient than downloading one large model at the start of video streaming. We now describe how the model is built in PARSEC.

Super-resolution architecture: Our architecture of super-resolution model is shown in Figure 2. We use a deep convolutional neural network (CNN) similar to [24], [45] to capture high level features in the video. We vary the number of network layers depending on the length of the video segment and the desired quality of the generated video. Each convolutional layer is followed by a LeakyRelu activation function [26], and Batch normalization for faster learning [19]. The neural network first extracts the high level features from low-level pixels and uses a non-linear mapping function to learn the original missing content details. Finally, the network uses a deconvolution layer to map the high resolution directly from the low resolution without image interpolation. More details of super-resolution can be found at [24], [31], [45].

Model training: Each model is trained for one video segment. First, the original video is divided into temporal segments (e.g., 1-2 secs) and tiled spatially (e.g., 192×192 pixels). Each tile is then down-scaled (e.g., 24×24 pixels) and compressed using a standard H.265 encoder, resulting in ULR tiles. Each ULR tile is then decoded and fed to the neural network as input to reconstruct original high resolution using the ground-truth tiles. While training, we use the PSNR metric [47] as the loss function to directly optimize for PSNR quality. We manually fine-tune the number of layers and filter size to achieve the desired median quality when mapped from ULR tiles to high resolution. We empirically determine optimal values for all the design choices such as tile size, the extent to which we down-scale and the length of the segment (as described in §V).

C. Benefits of Super-resolution versus HEVC Encoding

It is important to understand the quality of the reconstructed video in the super-resolution approach and its bandwidth usage. We compare the PSNRs of the generated video in the super-resolution approach vs. standard HEVC encoding at multiple bit rates (Figure 3(a)) and the corresponding actual demands on the network measured in bytes (Figure 3(b)). Clearly, the super-resolution approach performs better quality-wise than the 1 Mbps video while saving about 7× bandwidth (median). The savings are more significant at the 90^{th} percentile. This comprehensively demonstrates the potential of the super-resolution approach for streaming 360° videos.

IV. Neural-Aware ABR Algorithm

PARSEC's Adaptive Bit Rate (ABR) algorithm explores the tradeoff between available network and client-side compute capacities. In contrast to existing 360° video streaming solutions that use the network as their only resource, PARSEC uses a hybrid approach for its ABR: 1) it uses the available network capacity to stream a subset of tiles from the server; 2) it decides on the bitrate (quality) of the tiles to be fetched; 3) it also leverages the available compute capacity at the client device to 'generate' a different subset of tiles using the super-resolution technique. This latter step fetches the ULR tiles from the server plus the necessary DNN micro-models (§III).

Figure 4 shows the system architecture of PARSEC. The encoded video and ULR tiles are stored at the server. The client makes the decision about which tiles to fetch or generate using an ABR algorithm that is 'neural-aware,' i.e., understands the tradeoffs between the two methods. The decisions also take as input the user's viewport probability distribution and available network vs. compute capacity. The goal is to optimize the overall video quality of experience (QoE).

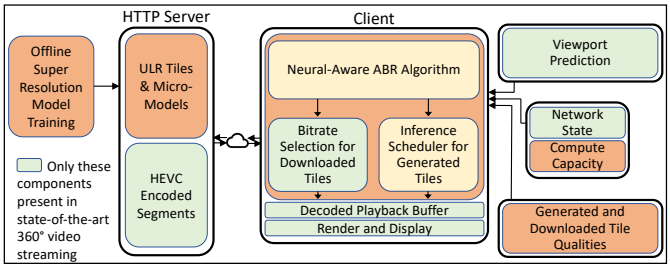


Fig. 4: PARSEC's end-to-end 360° video streaming system.

A. Modeling Quality of Experience

The 360° video is divided temporally into segments and spatially into N tiles. The ABR algorithm runs segment by segment and chooses a set of tiles to fetch from the server and another set of tiles to generate at the client for the segment being considered. For generating the tiles, the ULR representation of all tiles and the DNN micro-model are also downloaded from the server. It is important to note that not all tiles need to be fetched or generated.

Viewport prediction: A viewport prediction algorithm uses 1) offline analysis of video data and 2) user's (online) head tracking trace to predict the user's viewport at the playback time of the current segment being considered by the algorithm. There is a growing interest in such algorithms in literature [15], [23], [32], [34]. They essentially analyze the video to discover salient features of the scene that is likely to capture the viewer's attention. This is augmented with the head tracking trace (that captures past viewports) to estimate which portion of the scene the user is likely to view in future. Such estimates are typically generated as a probability distribution p_i over all tiles i comprising the 360° scene.

We follow a machine learning approach to predict the viewport. The approach is a more refined version of recent work [15]. The input to our prediction is saliency [13] and motion [44] maps of the video (extracted offline) combined with the online head movement data.

Video Quality of Experience (QoE): The video QoE is characterized by the quality of the part of the scene actually viewed during playback. This quality is influenced by the resolution of the video shown during playback. It is also influenced by missing video data (e.g., missing segments or tiles), which typically will cause the player to stall. The QoE formulation in PARSEC follows the traditional methods adopted in video streaming literature [30], [32], [46] except that PARSEC must take into account the quality of tiles generated versus tiles downloaded.

We number the individual tiles in a segment from 1 to N in row-major order. Let $r_{i,D}$, $r_{i,G}$ and $r_{i,M}$ denote binary decision variables that are set to 1(0) if i^{th} tile is (not) downloaded, (not) generated and (not) missed respectively. It is possible to download a tile at different quality levels, which we denote by $q_{i,D}$. The quality level q here is a function of the video bitrate R and indicative of the viewing quality during playback. This can be modeled in various ways [27], [46] such as simply using the bitrate directly, i.e., $q(R) = R$ or using it as an index to a table of possible rates R. We take the latter approach (also used in [32]) and model quality levels as an integer $0, \ldots, k$, with a larger number indicating a higher quality. Quality 0 indicates no playback or missing data. Finally, note that the quality of a generated tile $q_{i,G}$ is constant for a given tile but could vary between tiles.

We model the expected playback quality of a segment $E(Q)$ as the sum of expected quality of the individual tiles:

$$E(Q) = \sum_{i=1}^{N} p_i (q_{i,D} r_{i,D} + q_{i,G} r_{i,G}) \tag{1}$$

There is a loss of quality whenever a tile is missed. We represent this tile miss $E(M)$ as:

$$E(M) = \sum_{i=1}^{N} p_i r_{i,M} \tag{2}$$

There is also a loss of quality due to variations in the quality of the viewed tiles. This can be both due to changes in the quality across different segments, and across all tiles of the same segment. We utilize standard deviation V_s of quality of the tiles in a segment:

$$V_s = \sum_{i=1}^{N} \text{StdDev}[p_i (q_{i,D} r_{i,D} + q_{i,G} r_{i,G})] \tag{3}$$

We utilize the expected change in the value of quality (Q) of tiles across different segments:

$$V_t = |E(Q) - Q^{t-1}|, \tag{4}$$

where Q^{t-1} denotes the quality of the previous tile. Note that Q^{t-1} is a known quantity (not an expectation), since the actual decisions and experiences in the previous segment can be recorded. Thus, the overall quality of experience (QoE) is given by their linear combinations:

$$QoE = E(Q) - \beta E(M) - \xi (V_s + V_t), \tag{5}$$

where β and ξ represent the different weights attached to each component of QoE. We auto-tune β and ξ using a technique similar to Oboe [12]. Our objective is to maximize QoE in equation (5) while ensuring that we get a feasible solution.

Constraints: The following constraints ensure the feasibility of the solution. The quality due to download can only be positive if the tile is downloaded, i.e.

$$q_{i,D} \geq r_{i,D} \tag{6}$$

Also, every tile must be either downloaded, generated or missed, i.e.,

$$r_{i,D} + r_{i,G} + r_{i,M} = 1, \ \forall i = 1, \ldots, N \tag{7}$$

Finally, all generation and downloading must complete some time δ before the playback (P_t is the time until playback). Let $d(q_{i,D})$ be the time required to fetch tile i at quality $q_{i,D}$. Similarly, let $d(q_{i,G})$ be the time required to generate tile i. Then, this constraint is represented as:

$$\sum_{i=1}^{N} d(q_{i,D}) r_{i,D} + \delta \leq P_t, \quad \textbf{and} \quad \sum_{i=1}^{N} d(q_{i,G}) r_{i,G} + \delta \leq P_t \tag{8}$$

They represent network capacity and compute capacity constraints respectively.

Parameter	Description
N	Total number of tiles in a segment
p_i	View probability of tile i
$d(q_{i,D})$	Time to download tile i
$d(q_{i,G})$	Time to generate tile i
P_t	Time until start of playback
δ	Buffer period before playback time
$q_{i,D}$	Quality level of downloaded tile i
$q_{i,G}$	Quality level of generated tile i
$r_{i,X}$	Mutually exclusive decision variables, $X = D, G$ or M denoting download, generation or miss respectively

TABLE I: Parameters used in problem formulation.

B. Optimizing QoE

Our objective now is to maximize QoE in equation 5 for the segment being considered by the ABR algorithm, subject to the constraints in equations 6-8. While it is formulated as an ILP, we solve the optimization problem using a fast, greedy heuristic (§IV-C). The QoE (equation 5) is maximized subject to a set of constraints. The first two constraints, shown in equations 6 and 7 ensure that if a tile gets downloaded or generated, it is assigned a positive quality level. The last two constraints shown in equation 8 are capacity constraints ensuring that the download and generation of segments must be completed before playback. These constraints state that the time to get the tiles ready (from both generation and network download) should be no later than the playback time (P_t), with a delta (δ) time reserved for sundry computational work such as decoding of downloaded tiles and stitching of tiles to make scene ready for viewing. These constraints need to estimate available network bandwidth and available compute capacity. More on these shortly.

This optimization serves the purpose of ABR control. It relies on bandwidth estimation (constraint 8). While a variety of mechanisms could be adopted we use a technique similar to MPC [46], where available bandwidth is estimated using a harmonic mean predictor of past throughputs observed. The estimated throughput is used to estimate the download time of tiles based on their quality levels. The compute capacity estimation is relatively easier as the client device is typically not a shared resource and it has a well-defined fixed capacity.[5]

As observed in §III, the quality of these generated tiles can vary significantly. Hence choosing the tiles just based on tile probabilities may generate poor quality tiles. To address this, we extend the structure of the manifest file supported by DASH. As the server can generate the tiles offline, we calculate the quality of each tile generated and encode the quality information in the manifest file. This is analogous to encoding different representations of a tile in the manifest file, except here each generated tile has a fixed quality. We use this quality information in the ABR algorithm while scheduling tiles for network and compute.

The algorithm runs right before a segment starts playing to determine how the next segment will be downloaded and generated. Additionally, the ULR tiles and micro-model of the next segment are downloaded at the beginning of the current segment. The rate adaptation algorithm should be run frequently (i.e., need to choose small segments) because of the limited ability of viewport prediction. Hence the computation should be fast (few ms). Since the problem is NP-Hard, using an optimization solver does not guarantee a solution within such a limited period. We use a fast, greedy heuristic in our evaluation as described below.

[5]In this context, one can imagine use of edge computing to exploit more compute capacity on the client side. The same general formulation would still apply. We leave this as a future work.

C. Greedy Heuristic

Our greedy heuristic utilizes the fact that increasing the quality of a tile requires either additional network bandwidth or compute power, and thus it must be done carefully based on the view probability of tiles and ensuring satisfaction of constraints defined in (8). For brevity we only provide a high-level textual description below. The algorithm begins by setting quality level of all tiles to zero (indicating that they will be neither downloaded nor generated). The algorithm then increases the quality level of the tile with the highest probability by one (i.e., next possible quality). This may be either using compute or download, depending on the current quality level. In the next step, we check whether increasing the quality level of the same tile or the tile with the next highest probability leads to a higher value of QoE. We choose the option that provides a higher value. In this way, we keep selecting the better option until the constraints in (8) are reached or QoE can no longer be improved any further.

The time complexity of the algorithm is $O(N^2k)$, where k is the number of qualities available (analysis omitted for brevity). In the actual experiment reported later, the heuristic runs in less than 2ms for 200 tiles and 5 quality levels.

D. Rescheduling of Tile Generation

As noted before, viewport prediction is often not sufficiently accurate. While our viewport prediction is superior to Flare [32] over longer time horizons, this is still not enough (only 62% accurate over 3 sec). PARSEC is able to address this by recalculating the viewing probabilities of tiles for future frames *in the current segment being played*. This provides a much higher accuracy as the time horizon is now much shorter. The new information is used to reschedule tile generation which greatly reduces tile miss rate.

V. Evaluation

We implement the 360° video streaming system as shown in Figure 4 and evaluate the performance w.r.t. multiple approaches on a mobile client platform. Below we present the testbed, our methodology and then experimental results.

A. Testbed

Client and server implementation: The client video player is implemented in C language based on an open-source adaptive streaming video player, MP4Client [7]. We present results using Google Pixel2 as the client (Adreno 540 GPU). To generalize the results, we also evaluate on 5 additional devices

(Figure 9). We use Node.JS to host the content on the server using MPEG-DASH compliant HTTP adaptive streaming [36]. The server is hosted using NodeJS on a Linux Desktop.

Video segmentation: We use GPAC's MP4Box tool to divide the video spatially into segments and then temporally into tiles. We use `kvazaar` [6], an HEVC based implementation [38] for encoding the videos. Overall, the step-by-step procedure to prepare the DASH segments is 1) encode the videos using `kvazaar`, 2) divide the video spatially using MP4Box, 3) package the video in DASH format into tiled segments with multiple representations and 4) generate the manifest file.

Offline processing: For training DNN micro-models, we use Keras [4] and Tensorflow [10] in Python and an Nvidia GTX 1070 GPU. For each video, we get ULR tiles for each tile in a segment and one model for each segment. The training time for each video (i.e., for all micro-models) is around 20 minutes. Apart from learning the DNN models, the videos are also processed offline for traditional ABR streaming, by the DASH standard [36]. Once the offline processing is complete, the DASH segments, the micro-models, and the ULR tiles are stored in the server. From the server's point of view, these additional models and ULR tiles are simply treated as new content; it will be streamed to the client upon a request. The result is that there is no server-side modification and we can work with standard MPEG-DASH capable server.

We evaluate PARSEC under four different network conditions and compare it with four state-of-the-art alternatives. We describe our experimental methodology first.

B. Network Settings

End-to-end experiments over WiFi: We host a video server in two different locations: Loc1 is around $20ms$ RTT from the client and Loc2 which is $40ms$ RTT from the client. We choose these locations to serve as a proxy for CDNs. The client is hosted in our lab. We use Aruba WiFi AP with 802.11ac link speed. We stream the video from the server to the client according to the different streaming algorithms. We do not throttle the speed.

Real network traces: We collect real network traces from two popular sources—FCC released broadband dataset [8] and 4G/LTE network measurements from the Ghent university [5]. We filter the traces to have a minimum bandwidth of 1 Mbps to initiate the video flow. After filtering, FCC dataset has an average bandwidth of 8.2 Mbps with a standard deviation of 3.6 Mbps and Belgium dataset has an average of 19.3 Mbps with a standard deviation of 6.1 Mbps.

Synthetic traces: Finally, we conduct a small set of experiments on synthetic network conditions to stress test PARSEC under poor (e.g., 1 Mbps) versus good (e.g., 20 Mbps) network conditions. We use Mahimahi [28] to emulate the network conditions.

C. Experimental Methodology

We compare PARSEC with the following alternatives:

- **VP_only**: VP_only [15] uses viewport prediction to reduce bandwidth required to fetch 360° videos. It only fetches predicted viewport-specific tiles and suffers from frequent tile misses.
- **Flare**: Flare [32] is the state-of-the-art 360° video streaming system that combines viewport prediction with an ABR algorithm. Flare preemptively fetches some non-viewport tiles to compensate for the low accuracy of viewport prediction.
- **NAS-regular:** NAS [45] is designed for regular video streaming and leverages super-resolution. Because NAS is not designed for 360° video streaming, it does not use viewport prediction and fetches all tiles in the segment. We call it as NAS-regular.
- **NAS-360:** For a fair comparison, we also experiment with a version of NAS that is adapted for 360° video streaming. We call this scheme NAS-360. NAS-360 uses our viewport prediction to fetch only viewport-specific tiles. This essentially means NAS-360 is PARSEC with NAS's single model inference instead of our micro-model inference.

For a fair evaluation, we use the same viewport prediction algorithm (described in §IV) for all alternatives that use viewport prediction. We verify that our viewport prediction technique provides 26% better median accuracy than that described in Flare [32] for one second window.

Other regular video streaming techniques include BOLA [37], and Pensieve [27] perform significantly worse compared to PARSEC for 360° video streaming (because they are viewport agnostic). We omit the comparisons for these other systems in the interest of space.

360° video dataset: We use the most commonly used 360° video head movement dataset [25] in our evaluation. The dataset contains a total of 500 traces with 10 videos, each video watched by 50 users. Each trace contains a user's head position (yaw, pitch and roll) for every frame. Using the raw head movement data, we derive the viewport and viewport-specific tiles. The videos are typically about 1 min long. In the default case, we split each video temporally into 1 sec segments. Each video is encoded and projected using equi-rectangular projection at 4K quality (3840×1920). For ABR, we transcode the video in 5 different quality levels – 1 Mbps, 5 Mbps, 10 Mbps, 15 Mbps and 20 Mbps.

Performance metrics: We measure performance using three metrics — (i) quality level (as defined in §IV-A), (2) miss ratio – fraction of tiles that are not available in the user's viewport as determined by the user's head movement, and (3) QoE as defined in equation 5. We present QoE in a normalized form against the maximum QoE possible. In some evaluations where multiple network traces are used (e.g., Figure 5), the QoE for each segment is averaged over all traces (average QoE or normalized average QoE, as appropriate).

Parameter selection: There are several parameters involved in evaluating the performance. Through empirical analysis, we choose the following design decisions. We use a tile size of 192x192 to achieve efficient viewport adaptation with minimal cross-tile compression overheads. We down-scale the tiles to 24x24 to avoid extreme quality loss and large models. We aim to achieve at least 30dB PSNR which is a minimum

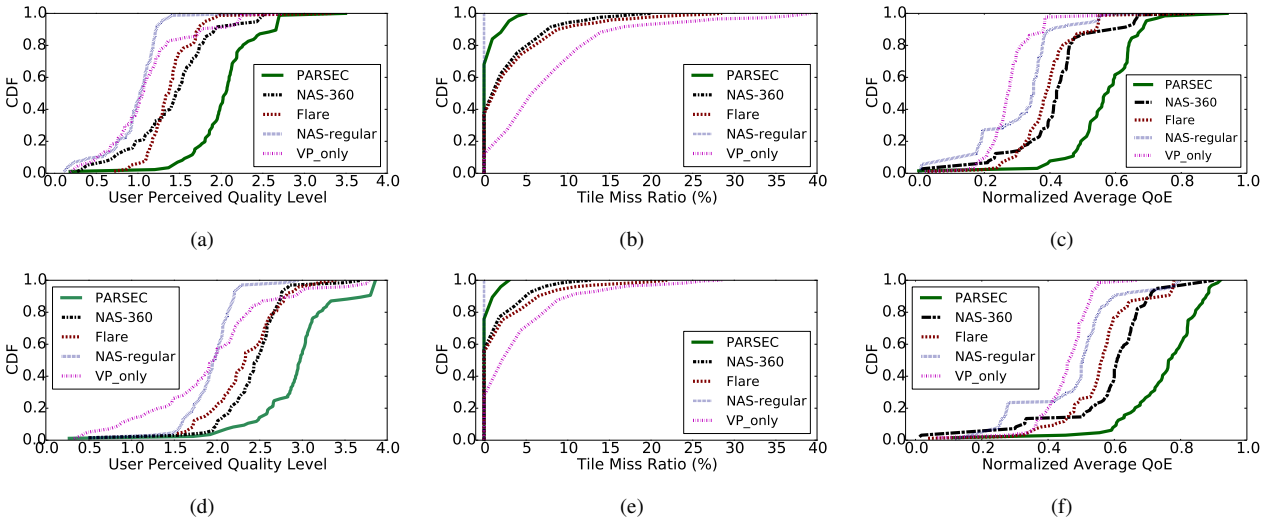


Fig. 5: Comparing PARSEC with state-of-the-art video streaming approaches. The experiments are done on a Google Pixel2 phone over real network traces from a broadband network released by the FCC (a-c) and 4G/LTE network from Belgium (d-f).

perceivable quality while inferring. We experiment with a segment duration of 1-3 sec because the prediction accuracy is very low beyond 3 sec.

D. Performance Results

Performance under broadband network traces: Figure 5 (a)-(c) shows the performance for the four alternatives over real broadband traces released by FCC. PARSEC improves performance by 61% and 48% on average quality level and normalized average QoE compared to the state-of-the-art 360° streaming protocol Flare. PARSEC exploits both client's compute and network resources, while Flare [32] only uses network resources to fetch video tiles.

PARSEC also outperforms NAS-360 by 42% in terms of QoE. Recall that NAS-360 is a version of NAS [45] that we adapted for 360° streaming with viewport prediction (this is a conservative estimate here because we eliminate model download for NAS-360). In the case of NAS-360, large model size results in poor inference rate and can only generate a few tiles per second on the device. NAS-regular performs even worse because it is viewport agnostic (i.e., brings in all tiles) and suffers from poor inference rate. Notice that NAS-regular has no misses because it fetches all tiles, but with lower quality. VP_only streams viewport-specific tiles only and hence experiences high miss ratio because of inaccurate viewport prediction, and has in general, the worst performance.

Performance under 4G/LTE network traces: Figure 5 (d)-(f) shows performance of PARSEC and the four alternatives for the 4G/LTE Belgium network traces. Under the 4G/LTE network traces, PARSEC improves performance by 30% and 37% of average quality level and normalized average QoE when compared with state-of-the-art system Flare [32].

The FCC's broadband network is more constrained compared to the 4G/LTE network traces in terms of capacity. As a result, the benefits of PARSEC are higher on the broadband network compared to the 4G/LTE network. All the other methods also have similar trends.

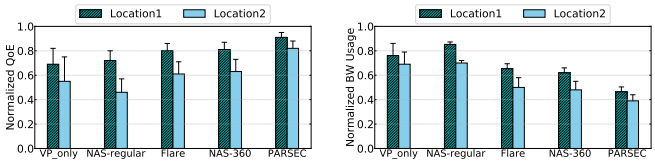


Fig. 6: QoE and bandwidth usage under real WiFi network with no bandwidth throttling (Pixel2 phone). The error bars represent the standard deviation.

End-to-end experiments over WiFi: We compare the performance of the four alternatives for a highly provisioned WiFi network setting with no throttling. These are end-to-end experiments where the videos are hosted in two server locations, Loc1 and Loc2. Figure 6a shows normalized average QoE. Similar to the experiments with network traces, PARSEC outperforms the four alternatives. PARSEC improves average QoE by 17% and 28% compared to Flare in Loc1 and Loc2 respectively. Loc1 is closer to the client with a lower RTT. This makes the network performance better, resulting in a lower overall improvement.

We also study the bandwidth utilization. Normalized bandwidth usage of PARSEC is 68% and 43% lower compared to NAS-regular and Flare respectively (Figure 6b). NAS-regular is viewport agnostic and streams all tiles. Flare streams some non-viewport tiles to compensate for misses in case the user's viewport changes. PARSEC has the lowest bandwidth usage because it chooses carefully which tiles to download at high resolution and which tiles to generate at the client. NAS-360 performs slightly better than Flare both in terms of QoE and bandwidth usage because some of the tiles will be enhanced at the client.

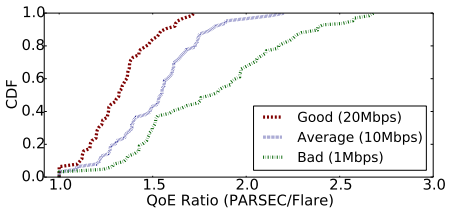
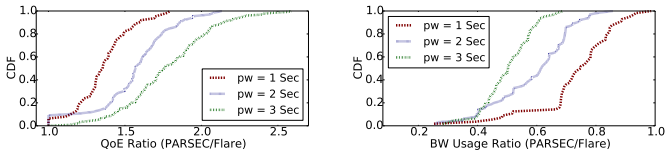


Fig. 7: Improvement in average QoE for PARSEC over Flare under good, average, and bad network conditions (Pixel2 phone).



(a) (b)

Fig. 8: QoE and bandwidth usage ratio for PARSEC over Flare for different video lengths (Pixel2 phone).

Different network conditions: We also evaluate the four alternatives under an ideal network scenario where both the client and server are in the same VLAN and the network capacity is high. PARSEC performs similar to NAS-regular in terms of QoE. However, PARSEC requires 2.3× and 1.8× less bandwidth compared to NAS-regular and Flare respectively (not shown here for brevity). This is because, when there are no network constraints, NAS-regular will fetch all tiles at high quality from the server.

PARSEC is designed specifically to work well when there is insufficient bandwidth. To stress test PARSEC, we compare the performance of PARSEC and Flare under three synthetic network scenarios—1 Mbps, 10 Mbps, and 20 Mbps average throughput. We only compare against Flare as it is the state-of-the-art end-to-end viewport adaptive system. Figure 7 shows the QoE ratio of PARSEC to Flare under these throughput rates. PARSEC sees more benefits under poorer networks. PARSEC outperforms Flare by 1.3×, 1.5×, and 1.8× under 20 Mbps, 10 Mbps and 1 Mbps respectively.

Streaming longer video segments We evaluate PARSEC to determine the benefits of supporting longer video segments. We use segments of duration 1s, 2s, and 3s, corresponding to a prediction window (pw) of 1s, 2s and 3s, respectively. Beyond 3s, the viewport prediction accuracy is not acceptable. These experiments were performed on the 4G/LTE traces collected from Belgium. Again, we only compare against Flare because it is the state-of-the-art viewport adaptive 360° video streaming system. Figure 8 shows normalized average QoE and network usage for three different segment durations. The key takeaway is that PARSEC can improve QoE by 1.7× while simultaneously minimizing the network usage by 47% compared to Flare for segment duration of 3s. Also, longer segments have better compression efficiency because of exploiting redundancy for longer duration. The QoE and bandwidth usage ratio both are improved substantially from

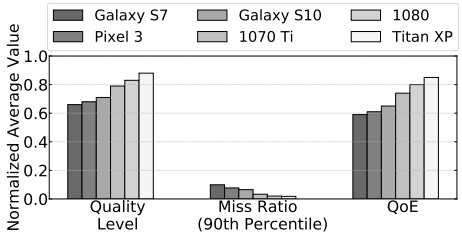


Fig. 9: Impact of GPU capacity on the QoE of PARSEC. Various mobile and desktop GPUs are considered. FCC traces are used for network. The Y-axis is the value of each metric normalized against it's maximum.

1s to 3s segments. We find that the dynamic rescheduling is playing crucial role for the 3s segments as the viewport prediction accuray is poor (less than 65%), and this impacts Flare much more adversely than PARSEC.

Impact of compute capacity: We evaluate PARSEC's performance for varying compute capacity by experimenting with six different devices (Figure 9). We specifically include high-end Desktop-grade GPUs as we expect future smartphones will have such powerful GPUs. Nvidia Titan XP is the highest-performing GPU used, while Galaxy S7 (Adreno 530 GPU) is the weakest. Figure 9 shows that as we increase the compute capacity from Galaxy S7 to Titan XP, the normalized quality level and QoE are increased by 31% and 44% respectively. The 90^{th} percentile miss ratio also decreases by 7%. The key takeway is that PARSEC performs better with faster GPUs.

As a final note, PARSEC does have additional energy overhead – varying between 12-22% in our evaluations over other streaming methods (as measured using Snapdragon Profiler [3]). This is to be expected due to the additional GPU use. We expect that advances in low-power GPU technologies for mobile platforms will address this problem.

VI. Conclusions

We have described PARSEC, a system that combines network and compute resources intelligently to stream 360° videos with high quality. PARSEC leverages the super-resolution technique that involves compressing the video at the server and then running deep learning inference at the client to enhance the video to high resolution. Since the deep learning model sizes can be big for 360° videos, PARSEC uses micro-models over short video segments to reduce model size and inference time. PARSEC then intelligently combines the super-resolution technique with traditional video encoding techniques so that the system can exploit the advantages of both by combining network and compute resources. PARSEC outperforms the state-of-the-art 360° video streaming systems under various network conditions.

Acknowledgements

This work was partially supported by NSF grants CNS-1642965 and CNS-1718014.

REFERENCES

[1] http://mathworld.wolfram.com/EquirectangularProjection.html.

[2] https://wiki.panotools.org/Cubic_Projection.

[3] https://developer.qualcomm.com/sites/default/files/docs/snpe/.

[4] Chollet, françois and others, keras. https://keras.io/, 2015.

[5] 4G/LTE bandwdith logs, Ghent University, Belgium. https://users.ugent.be/ jvdrhoof/dataset-4g/, 2016.

[6] Kvazaar. https://github.com/ultravideo/kvazaar, 2017.

[7] GPAC. https://github.com/gpac/gpac, 2017.

[8] Measuring broadband america, FCC. https://www.fcc.gov/reports-research/reports/measuring-broadband-america/raw-data-measuring-broadband-america-eighth, 2018.

[9] https://www.androidauthority.com/qualcomm-snapdragon-mobile-npu-896223/, 2019.

[10] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: a system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

[11] Shahryar Afzal, Jiasi Chen, and KK Ramakrishnan. Characterization of 360-degree videos. In *Workshop on VR/AR Network*, pages 1–6. ACM, 2017.

[12] Zahaib Akhtar, Yun Seong Nam, Ramesh Govindan, Sanjay Rao, Jessica Chen, Ethan Katz-Bassett, Bruno Ribeiro, Jibin Zhan, and Hui Zhang. Oboe: auto-tuning video abr algorithms to network conditions. In *SIGCOMM*, pages 44–58. ACM, 2018.

[13] Ali Borji, Ming-Ming Cheng, Qibin Hou, Huaizu Jiang, and Jia Li. Salient object detection: A survey. *arXiv preprint arXiv:1411.5878*, 2014.

[14] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Image super-resolution using deep convolutional networks. *IEEE transactions on pattern analysis and machine intelligence*, 38(2):295–307, 2016.

[15] Ching-Ling Fan, Jean Lee, Wen-Chih Lo, Chun-Ying Huang, Kuan-Ta Chen, and Cheng-Hsin Hsu. Fixation prediction for 360 video streaming in head-mounted virtual reality. In *NOSSDAV*. ACM, 2017.

[16] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *NIPS*, pages 2672–2680, 2014.

[17] Mario Graf, Christian Timmerer, and Christopher Mueller. Towards bandwidth efficient adaptive streaming of omnidirectional video over http: Design, implementation, and evaluation. In *MMSys*. ACM, 2017.

[18] Pan Hu, Rakesh Misra, and Sachin Katti. Dejavu: Enhancing video-conferencing with prior knowledge. In *HotMobile*, pages 63–68. ACM, 2019.

[19] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[20] Tilke Judd, Krista Ehinger, Frédo Durand, and Antonio Torralba. Learning to predict where humans look. In *CVPR*, pages 2106–2113. IEEE, 2009.

[21] Jean Le Feuvre, Cyril Concolato, and Jean-Claude Moissinac. Gpac: open source multimedia framework. In *Multimedia Conference*, pages 1009–1012. ACM, 2007.

[22] Royson Lee, Stylianos I Venieris, Lukasz Dudziak, Sourav Bhattacharya, and Nicholas D Lane. Mobisr: Efficient on-device super-resolution through heterogeneous mobile processors. In *MobiCom*, pages 1–16, 2019.

[23] Chenge Li, Weixi Zhang, Yong Liu, and Yao Wang. Very long term field of view prediction for 360-degree video streaming. *arXiv preprint arXiv:1902.01439*, 2019.

[24] Bee Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, and Kyoung Mu Lee. Enhanced deep residual networks for single image super-resolution. In *IEEE CVPR Workshops*, pages 136–144, 2017.

[25] Wen-Chih Lo, Ching-Ling Fan, Jean Lee, Chun-Ying Huang, Kuan-Ta Chen, and Cheng-Hsin Hsu. 360 video viewing dataset in head-mounted virtual reality. In *MMSys*, pages 211–216. ACM, 2017.

[26] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3, 2013.

[27] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. Neural adaptive video streaming with pensieve. In *SIGCOMM*, pages 197–210. ACM, 2017.

[28] Ravi Netravali, Anirudh Sivaraman, Somak Das, Ameesh Goyal, Keith Winstein, James Mickens, and Hari Balakrishnan. Mahimahi: Accurate record-and-replay for {HTTP}. In *ATC*, pages 417–429, 2015.

[29] Anh Nguyen, Zhisheng Yan, and Klara Nahrstedt. Your attention is unique: Detecting 360-degree video saliency in head-mounted display for head movement prediction. In *Multimedia Conference*, pages 1190–1198. ACM, 2018.

[30] S. Park, A. Bhattacharya, Z. Yang, M. Dasari, S. R. Das, and D. Samaras. Advancing user quality of experience in 360-degree video streaming. In *2019 IFIP Networking Conference*, pages 1–9, May 2019.

[31] Sung Cheol Park, Min Kyu Park, and Moon Gi Kang. Super-resolution image reconstruction: a technical overview. *IEEE signal processing magazine*, 20(3):21–36, 2003.

[32] Feng Qian, Bo Han, Qingyang Xiao, and Vijay Gopalakrishnan. Flare: Practical viewport-adaptive 360-degree video streaming for mobile devices. In *MobiCom*, pages 99–114. ACM, 2018.

[33] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[34] Yago Sanchez, Gurdeep Singh Bhullar, Robert Skupin, Cornelius Hellge, and Thomas Schierl. Delay impact on MPEG OMAFs tile-based viewport-dependent 360 video streaming. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2019.

[35] Wenzhe Shi, Jose Caballero, Christian Ledig, Xiahai Zhuang, Wenjia Bai, Antonio de Marvao, Tim Dawes, Declan ORegan, and Daniel Rueckert. Cardiac image super-resolution with global correspondence using multi-atlas patchmatch. In *MICCAI*, pages 9–16. Springer, 2013.

[36] Iraj Sodagar. The MPEG-DASH standard for multimedia streaming over the internet. *IEEE MultiMedia*, (4), 2011.

[37] Kevin Spiteri, Rahul Urgaonkar, and Ramesh K Sitaraman. Bola: Near-optimal bitrate adaptation for online videos. In *INFOCOM*, pages 1–9. IEEE, 2016.

[38] Gary J Sullivan, Jens-Rainer Ohm, Woo-Jin Han, and Thomas Wiegand. Overview of the high efficiency video coding (HEVC) standard. *IEEE Transactions on circuits and systems for video technology*, 22(12):1649–1668, 2012.

[39] Ron Tenne, Uri Rossman, Batel Rephael, Yonatan Israel, Alexander Krupinski-Ptaszek, Radek Lapkiewicz, Yaron Silberberg, and Dan Oron. Super-resolution enhancement by quantum image scanning microscopy. *Nature Photonics*, 13(2):116, 2019.

[40] Nikolaos Thomos, Nikolaos V Boulgouris, and Michael G Strintzis. Optimized transmission of jpeg2000 streams over wireless channels. *IEEE Transactions on image processing*, 15(1):54–67, 2005.

[41] Mengbai Xiao, Chao Zhou, Yao Liu, and Songqing Chen. Optile: Toward optimal tiling in 360-degree video streaming. In *Multimedia Conference*, pages 708–716. ACM, 2017.

[42] Mengbai Xiao, Chao Zhou, Viswanathan Swaminathan, Yao Liu, and Songqing Chen. Exploring spatial and temporal adaptability in 360-degree videos over http/2. In *INFOCOM*, pages 953–961. IEEE, 2018.

[43] Lan Xie, Zhimin Xu, Yixuan Ban, Xinggong Zhang, and Zongming Guo. Improving qoe of 360 video streaming using tile-based http adaptive streaming. In *Multimedia Conference*, pages 315–323. ACM, 2017.

[44] Xiaodong Yang, Chenyang Zhang, and YingLi Tian. Recognizing actions using depth motion maps-based histograms of oriented gradients. In *Multimedia Conference*, pages 1057–1060. ACM, 2012.

[45] Hyunho Yeo, Youngmok Jung, Jaehong Kim, Jinwoo Shin, and Dongsu Han. Neural adaptive content-aware internet video delivery. In *OSDI*, pages 645–661, 2018.

[46] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. A control-theoretic approach for dynamic adaptive video streaming over http. In *ACM SIGCOMM*, volume 45, pages 325–338. ACM, 2015.

[47] Matt Yu, Haricharan Lakshman, and Bernd Girod. A framework to evaluate omnidirectional video coding schemes. In *2015 IEEE International Symposium on Mixed and Augmented Reality*, pages 31–36. IEEE, 2015.

[48] Alireza Zare, Alireza Aminlou, Miska M Hannuksela, and Moncef Gabbouj. HEVC-compliant tile-based streaming of panoramic video for virtual reality applications. In *Multimedia Conference*, pages 601–605. ACM, 2016.

[49] Chao Zhou, Zhenhua Li, and Yao Liu. A measurement study of oculus 360 degree video streaming. In *MMSys*, pages 27–37. ACM, 2017.

[50] Wilman WW Zou and Pong C Yuen. Very low resolution face recognition problem. *IEEE Transactions on image processing*, 21(1):327–340, 2012.