Energy-Efficient Video Processing for Virtual Reality

Yue Leng*
UIUC
yueleng2@illinois.edu

Chi-Chun Chen*
University of Rochester
cchen120@ur.rochester.edu

Qiuyue Sun
University of Rochester
qsun15@u.rochester.edu

Jian Huang
UIUC
jianh@illinois.edu

Yuhao Zhu
University of Rochester
yzhu@rochester.edu

Abstract

Virtual reality (VR) has huge potential to enable radically new applications, behind which spherical panoramic video processing is one of the backbone techniques. However, current VR systems reuse the techniques designed for processing conventional planar videos, resulting in significant energy inefficiencies. Our characterizations show that operations that are unique to processing 360° VR content constitute 40% of the total processing energy consumption.

We present EVR, an end-to-end system for energy-efficient VR video processing. EVR recognizes that the major contributor to the VR tax is the projective transformation (PT) operations. EVR mitigates the overhead of PT through two key techniques: semantic-aware streaming (SAS) on the server and hardware-accelerated rendering (HAR) on the client device. EVR uses SAS to reduce the chances of executing projective transformation on VR devices by pre-rendering 360° frames in the cloud. Different from conventional pre-rendering techniques, SAS exploits the key semantic information inherent in VR content that is previously ignored. Complementary to SAS, HAR mitigates the energy overhead of on-device rendering through a new hardware accelerator that is specialized for projective transformation. We implement an EVR prototype on an Amazon AWS server instance and a NVIDA Jetson TX2 board combined with a Xilinx Zynq-7000 FPGA. Real system measurements show that EVR reduces the energy of VR rendering by up to 58%, which translates to up to 42% energy saving for VR devices.

CCS Concepts

• **Hardware → Hardware accelerators**; **Emerging architectures**; *Platform power issues.*

Keywords

virtual reality, video processing, hardware accelerator, projective transformation, pre-rendering, energy efficiency

[*] Yue Leng and Chi-Chun Chen are co-primary authors.

ISCA '19, June 22–26, 2019, Phoenix, AZ, USA
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6669-4/19/06...$15.00
https://doi.org/10.1145/3307650.3322264

ACM Reference Format:
Yue Leng, Chi-Chun Chen, Qiuyue Sun, Jian Huang, and Yuhao Zhu. 2019. Energy-Efficient Video Processing for Virtual Reality. In *The 46th Annual International Symposium on Computer Architecture (ISCA '19), June 22–26, 2019, Phoenix, AZ, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3307650.3322264

1 Introduction

Virtual Reality (VR) has profound social impact in transformative ways. For instance, immersive VR experience is shown to reduce patient pain [8] more effectively than traditional medical treatments, and is seen as a promising solution to the opioid epidemic [15]. One of the key use-cases of VR is 360° video processing. Unlike conventional planar videos, 360° videos embed panoramic views of the scene. As users change the viewing angle, the VR device renders different parts of the scene, mostly on a head-mounted display (HMD), providing an immersive experience. 360° videos have gained tremendous momentum recently with the sales of 360° cameras expected to grow by 1500% in a few years [69], new compression standard being discussed and established [11], and companies such as YouTube and Facebook heavily investing on it.

A major challenge in VR video processing today is the excessive power consumption of VR devices. Our measurements show that rendering 720p VR videos in 30 frames per second (FPS) consistently consumes about 5W of power, which is twice as much power than rendering conventional planar videos and exceeds the Thermal Design Point (TDP) of typical mobile devices [32]. The device power requirement will only grow as users demand higher frame-rate and resolution, presenting a practical challenge to the energy-and-thermal constrained mobile VR devices.

The excessive device power is mainly attributed to the fundamental mis-match between today's VR system design philosophy and the nature of VR videos. Today's VR video systems are designed to reuse well-established techniques designed for conventional planar videos [19, 30, 37]. This strategy accelerates the deployment of VR videos, but causes significant energy overhead. More specifically, VR videos are streamed and processed as conventional planar videos. As a result, once on-device, each VR frame goes through a sequence of spherical-planar projective transformations (PT) that correctly render a user's current viewing area on the display. The PT operations are pure overhead uniquely associated with processing VR videos – operations that we dub "VR tax." Our characterizations show that "VR tax" is responsible for about 40% of the processing energy consumption, a lucrative target for optimizations.

This paper proposes EVR, an end-to-end energy-efficient VR system. It consists of two complementary optimization primitives:

semantic-aware streaming and hardware-accelerated rendering. EVR synergistically combines the two techniques and achieves significant energy reduction for VR devices.

Specifically, semantic-aware streaming (SAS) removes the VR-induced operations from VR devices as much as possible. This is achieved by tasking the streaming server with the projective transformation operations. Effectively, the streaming server pre-renders the pixels falling within the user's viewing area and stream only those pixels. In this way, the VR video can be directly displayed on-device in the same way as a conventional planar video, greatly reducing the overall energy consumption.

Although pre-rendering has been exploited in various contexts [1, 23, 47], pre-rendering VR content poses unique challenges. Specifically, it requires us to predict user's viewing area in every frame. Ideally, we should do so without incurring extra client overhead. Recent work that uses machine learning techniques on the VR device to predict user's head orientation [58, 63] potentially increases the device power. Instead, we propose an effective and lightweight prediction solution. We observe that user's viewing area can be accurately predicted by object semantics information inherent in VR content that is previously unrecoginized. Specifically, our characteristic study of VR video workloads reveals that VR users in the steady state tend to track the same set of objects across frames. Thus, we propose a first-order viewing area predictor based on object trajectories within a VR video.

An important implication of semantic-aware prediction is that it might mis-predict in scenarios where user do not track objects, in which cases rendering VR videos on-device is inevitable. We observe that the major inefficiency in on-device VR video rendering is that VR rendering systems cast the PT operations as a texture mapping problem, which is then delegated to GPUs to maximize the reuse of the existing hardware. Although well-optimized for generic texture mapping, GPUs incur high energy overhead for the specific texture mapping instance of VR projective transformations.

To mitigate the energy overhead of on-device rendering, we propose hardware-accelerated rendering (HAR), which incorporates a new hardware accelerator specialized for PT. The hardware accelerator exploits the unique compute characteristics of PT to significantly reduce the energy consumption of VR video rendering, while maintaining just enough configurability to adapt to different transformation methods for ensuring general applicability [25].

Building on top of the two optimizing primitives – semantic-aware streaming and hardware-accelerated rendering, we design EVR, an end-to-end VR video system that optimizes for energy-efficiency. EVR includes a cloud component and a client component. The cloud component extracts object semantics from VR videos upon ingestion, and pre-renders a set of miniature videos that contain only the user viewing areas and that could be directly rendered as planar videos by leveraging the powerful computating resources on the cloud. The client component retrieves the miniature video with object semantics, and leverages the specialized accelerator for energy-efficient on-device rendering if the original full video is required. For VR applications whose content comes from panoramic videos available on the VR devices, the HAR can accelerate the video rendering with lower energy overhead.

We implement EVR in a prototype system, where the cloud service is hosted on an AWS instance while the client is deployed on a customize platform that combines the NVIDIA TX2 and Xilinx Zynq-7000 development boards which can represent a typical VR client device. We evaluate EVR on a wide variety of VR use-cases. Based on real system measurements on varius real-world VR datasets, EVR reduces the energy of VR rendering by up to 58%, which translates to up to 42% device-level energy savings. Overall, this paper makes the following contributions:

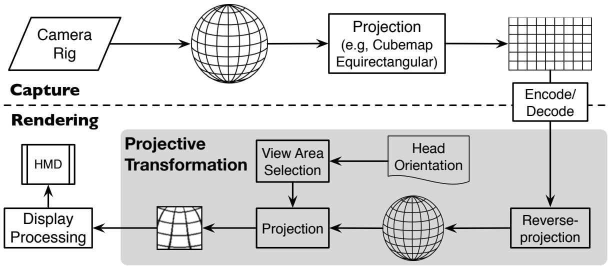


Figure 1: Illustration of today's VR video system.

- We provide a detailed power characterization of VR client devices and show that projective transformation is a significant source of the energy overhead uniquely introduced by VR video processing.
- We quantitatively demonstrate, based on the characteristic study of various VR video workloads, that VR users' viewing behaviors strongly correlate with visual object movements in VR content.
- We propose semantic-aware streaming that extracts visual object semantics from VR content and pre-renders VR content in the server to save energy for client devices.
- We design a hardware accelerator specialized for energy-efficient on-device VR video rendering.
- We design and implement EVR, which combines semantic-aware streaming and hardware-accelerated rendering. Measurements on a real-world cloud-client setup demonstrates significant reductions in VR device energy consumption.

The rest of this paper is organized as follows. § 2 discusses the background of 360° video rendering. § 3 presents our power characterization of VR devices. We present the overview of EVR in § 4. The designs of EVR's core components SAS and HAR are detailed in § 5 and § 6, respectively. We describe the implementation of EVR in § 7 and evaluate in § 8. We discuss the related work in § 9 and conclude in § 10.

2 VR Video System Background

Different from conventional planar video content, 360° VR content provides an immersive experience by encoding panoramic views in video frames. This section defines the scope of VR system that we study in this paper, and introduce the necessary background.

VR System Overview. A VR system involves two distinct stages: capture and rendering. Figure 1 illustrates a typical VR system today. VR videos are captured by special cameras [4, 21], which generate 360° images that are best presented in the spherical format. The spherical images are then projected to planar frames through one of the spherical-to-planar projections such as the equirectangular projection [27]. The planar video is either directly live-streamed to

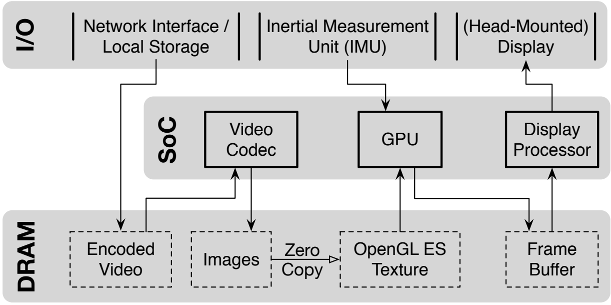
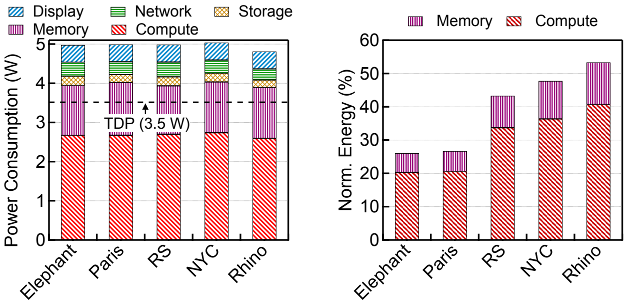


Figure 2: System architecture of a typical VR client device.



(a) Power distribution across the major components in a VR device.

(b) Contribution of the PT operations to the compute and memory energy.

Figure 3: Power and energy characterizations of VR device.

client devices for rendering (e.g., broadcasting a sports event) [61], or published to a content provider such as YouTube or Facebook and then streamed to client devices upon requests. Alternatively, the streamed videos can also be persisted in the local storage on a client device for future playback.

This paper focuses on client-side VR content rendering, i.e., after a VR video is captured, because rendering directly impacts VR devices' energy-efficiency. Capturing VR content requires hardware and software different from rendering, and is an active research area on its own. We refer interested readers to § 9 for more discussions.

Client-Side Rendering. In conventional planar video rendering, each frame could be directly displayed once decoded. However, each VR frame must go through a set of projective transformation (PT) operations before displayed on the HMD.

We demonstrate the procedure of PT in Figure 1. Conceptually, PT works as follows. A planar VR frame is first reversely projected back to the spherical format, from which the client player generates a spherical region that corresponds to the user's current viewing angle, which is dictated by the head orientation. The spherical region is then projected to the planar format again for display. It is worth emphasizing that in the context of 360° video rendering only *rotational* head motion is considered, and *translational* motion is not. This is different from other VR use-cases such as gaming where both rotational and translation movements are captured.

The reason that the PT operations are required at the rendering time is because 360° frames, after being captured, are projected from the spherical format to the planar format. The spherical-to-planar (S2P) projection might initially seem wasteful since VR clients have to perform a reverse projection (i.e., PT). In practice, however, S2P allows the compression and streaming techniques that have been well-optimized for planar videos to be used. This design strategy accelerates the adoption of VR videos, but significantly increases the VR device's compute overhead as quantified later.

It is worth noting that the size of the current viewing area is typically much smaller than the full sphere. Specifically, the viewing area size is characterized by an HMD's field-of-view (FOV), which captures the degrees of horizontal and vertical viewing angles. For instance, under an FOV of $120^\circ \times 90^\circ$, the viewing area is one-sixth of the full sphere. Off-the-shelf VR delivery systems stream entire frames because the user viewing area is typically unknown beforehand, leading to wasted resources. Recent proposals use view-guided streaming [26, 29] to predict user head movement and reduce the resolution of out-of-sight areas in a frame. These approaches, however, do not optimize energy consumptions because they still require the PT operations on VR client devices.

VR Device Hardware Architecture. VR devices fall into two categories: ones that are tethered with a base computing device such as a smartphone (e.g., Samsung Gear VR, Google Daydream), and standalone un-tethered devices with the computing device built-in (e.g., Oculus Go). Albeit the difference, the hardware architectures in both cases are similar because they both perform similar tasks and conform to similar form factors. Our work focuses on the underlying hardware and does not depend on a particular VR headset.

We show the block diagram of a typical hardware architecture of a VR headset in Figure 2. Central to the hardware is the Systems-on-a-chip (SoC), which interacts with I/O devices and the DRAM. The SoC includes three main Intellectual Property (IP) blocks key to VR video processing: Video Codec, GPU, and Display Processor, among which GPU is the major power consumer.

In conventional planar video rendering, the GPU can be bypassed. That is, video frames once decoded can be directly passed to the Display Processor, which performs necessary pixel manipulations (e.g., color-space conversion, rotation), and scans out the pixels to the display [2]. However, the GPU is crucial to rendering VR videos because the PT operations are executed on the GPU [19, 30, 37]. In fact, 360° rendering is a key workload that GPU IP vendors today tune their implementations for [19].

The rationale behind tasking the GPU with PT operations is that the latter can be naturally casted as texture mapping [38, 39], which is inherently supported by modern GPUs through the Texture Mapping Unit (TMU). Effectively, a planar frame is treated as a texture map, which is mapped to the spherical geometry by the client software (mostly an OpenGL ES program) and gets sampled according to the user head orientation provided by the IMU.

3 Energy Characterizations

Rendering VR videos consumes excessive power on the VR device, which is particularly problematic as VR devices are energy and thermal constrained. This section characterizes the energy consumption of VR devices. Although there are many prior studies that focused on energy measurement of mobile devices such as smartphones and smartwatches, this is the first such study that

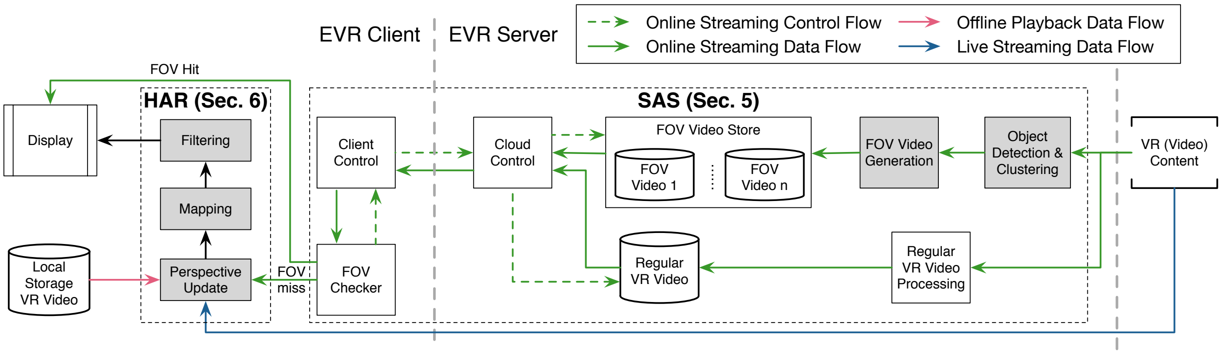


Figure 4: EVR overview. EVR consists of two synergistic components: SAS and HAR. Key augmentations to the existing VR system are shaded. EVR supports all three major VR use-cases: online streaming, live streaming, and offline playback.

specifically focuses on VR devices. We show that the energy profiles between VR devices and traditional mobile devices are different.

We conduct studies on a recently published VR video dataset, which consists of head movement traces from 59 real users viewing different 360° VR videos on YouTube [25]. We replay the traces to mimic realistic VR viewing behaviors. We assemble a custom VR device based on the NVIDIA Jetson TX2 development board [9, 13] in order to conduct fine-grained power measurements on the hardware, which are infeasible in off-the-shelf VR devices. We refer readers to § 8.1 for a complete experimental setup.

Power and Energy Distribution. We breakdown the device power consumption into five major components: display, network (WiFi), storage (eMMC), memory (DRAM), and compute (SoC). The storage system is involved mainly for temporary caching. We show the power distribution across the five components for the five VR video workloads in Figure 3a. The power consumption is averaged across the entire viewing period. Thus, the power consumption of each component is proportional to its energy consumption.

We make two important observations. First, the device consistently draws a power consumption of about 5W across all five VR videos. As a comparison, the Thermal Design Point (TDP) of a mobile device, i.e., the power that the cooling system is designed to sustainably dissipate, is around 3.5 W [32], clearly indicating the need to reduce power consumption.

Second, unlike traditional smartphone and smartwatch applications where network, display, and storage consume significant energy [24, 41, 50, 67], the energy consumptions of the three components in a VR device are relatively insignificant, contributing to only about 9%, 7%, and 4% of the total energy consumption, respectively. This indicates that optimizing network, display, and storage would lead to marginal energy reductions. More lucrative energy reductions come from optimizing compute and memory.

Contribution of VR Operations. We further find that energy consumed by executing operations that are uniquely associated with processing VR videos constitutes a significant portion of the compute and memory energy. Such operations mainly consist of PT operations as described in § 2. We show the energy contribution of the PT operations to the total compute and memory energy as a stacked bar chart in Figure 3b. On average, projective transformation contributes to about 40% of the total compute and memory energy, and is up to 53% in the case of video `Rhino`. The PT operations exercise the SoC more than the DRAM as is evident in their higher contributions to compute energy than memory energy.

Overall, our results show that the projective transformations would be an ideal candidate for energy optimizations.

4 EVR Overview

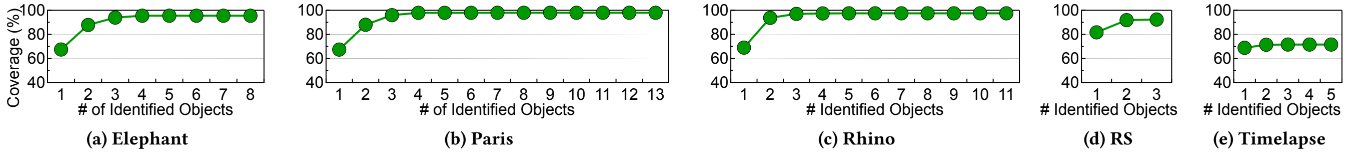
The goal of EVR is to reduce energy consumption of VR devices by optimizing the core components that contribute to the "VR tax." We present an end-to-end energy-efficient VR system with optimization techniques distributed across the cloud server and the VR client device as shown in Figure 4.

We first propose *Semantic-Aware Streaming* (SAS, § 5), which pre-renders the VR video in the cloud and thus avoids the energy-intensive PT operations on VR devices. SAS is developed upon an intuitive observation that VR users tend to track the same set of objects across frames. Thus, VR videos can be pre-rendered in the cloud by tracking object trajectories.

SAS would ideally remove the PT operations from the VR client completely. This, however, is almost impossible for two reasons. First, it is hard to continuously predict a user's viewing area in practice. Although users tend to track objects in stable states as confirmed by our study in § 5.1, they could also randomly orient the heads to explore the scene, resulting in mispredictions. Second, there are key VR use-cases where the VR videos do not go through a cloud server (e.g., offline playback), and thus cannot leverage SAS. Under such circumstances, on-device rendering is inevitable.

Thus, we propose *Hardware-Accelerated Rendering* (HAR, § 6), which provides specialized hardware support for energy-efficient PT operations. HAR exploits the unique compute and data access patterns of the PT algorithm while adapting to different algorithm variants. It thus strikes a balance between efficiency and flexibility.

SAS and HAR have different trade-offs. On one hand, HAR is applicable regardless where the VR videos are from, but does not completely remove the overhead of the PT operation. SAS potentially removes the PT operation altogether, but relies on that the VR video is published to a cloud server first. We now present SAS and HAR separately and describe how they are synergistically integrated. We then evaluate different variants of EVR that make



(a) Elephant (b) Paris (c) Rhino (d) RS (e) Timelapse

Figure 5: Each <x, y> point represents the percentage of frames (y) in which at least one of the x identified/detected objects appears in users' viewing area. Results indicate that users' attention centers around visual objects in VR content.

different uses of the two techniques. We show that combining the two, when applicable, achieves the best energy-efficiency.

5 Semantics-Aware Streaming

SAS pre-renders VR videos in the cloud in order to avoid the energy-intensive PT operations on the VR device. Pre-rendering requires predicting users' viewing area at every frame. This section presents a *server-only* solution that has little cost to the client devices.

We first use a real VR user dataset study to show that exploiting VR content semantics, especially visual object, offers an opportunity for pre-rendering VR videos in the cloud (§ 5.1). We then provide an SAS overview and describe its design principles (§ 5.2). After that, we discuss the detailed cloud architecture (§ 5.3) and the necessary support on the client (§ 5.4).

5.1 Object-Oriented Viewing Behaviors

Our key idea is to leverage video-inherent semantic information that is largely ignored by today's VR servers. SAS specifically focuses on one particular form of semantic information: visual object. We show that users tend to focus on objects in VR content, and object trajectories provide a proxy for predicting user viewing areas.

We leverage a recently published VR video dataset [25], which consists of head movement traces from 59 real users viewing different 360° VR videos on YouTube. We annotate the objects in the dataset and quantify the correlation between users' viewing area and the visual objects in the VR content in Figure 5.

We reach two key conclusions. First, users tend to pay attention to objects in VR videos. Even if only one object is detected in the video, users' viewing areas cover that single object in at least 60%, up to 80% (RS), of the frames. As the number of detected objects increases, the percentage of frames in which users focus on the detected objects increases to at least 80%, and reaches almost 100% in cases of Elephant, Paris, and Rhino. This indicates that frame areas that contain visual objects are likely to be watched by end-users, and thus pre-rendering and streaming those frame areas will likely satisfy users' needs.

We further confirm that users track the same set of objects across frame rather than frequently switching objects. Specifically, we measure the time durations during which users keep tracking the movement of the same object, and show the results in Figure 6 as a cumulative distribution plot. Each <x, y> point in the figure denotes the percentage of time (y) during which users track an object for at least a particular time duration (x). On average, users spend about 47% of time tracking an object for at least 5 seconds.

Second, the near 100% frame coverage in many videos as the number of identified objects increases indicates that the server can effectively predict user viewing area solely based on the visual objects without sophisticated client-side mechanisms such as using machine learning models to predict users' head movement [36, 58]. This observation frees the resource-constrained VR clients from performing additional work and simplifies the client design.

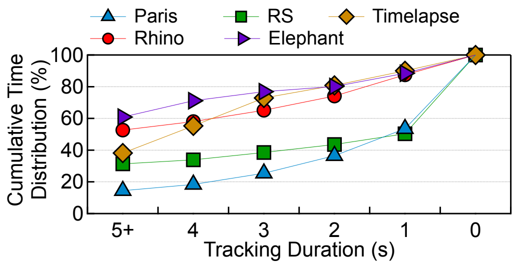


Figure 6: Cumulative distribution of tracking durations.

5.2 SAS Framework Overview

SAS has two major components: (1) a static and offline analysis component that extracts objects from the VR video upon ingestion and generates a set of FOV videos that could be directly visualized once on a VR device; (2) a dynamic and runtime serving component that streams FOV videos on demand to the VR device.

The static component generates FOV videos, each of which corresponds to the trajectory of a group of objects. Critically, the static component converts each FOV video frame from the spherical format to the planar format such that it could be directly visualized once on a client device, bypassing the energy-intensive projective transformation operations.

The dynamic component, at runtime, streams FOV videos to the client. We augment the new FOV video with metadata that corresponds to the head orientation for each frame. Once the FOV video together with its associated metadata is on the client side and before a FOV frame is sent to the display, the VR client compares the desired viewing area indicated by the head motion sensor with the metadata associated with the frame. If the two match, i.e., a *FOV-hit*, the client directly visualizes the frame on the display, bypassing the PT operations. Otherwise, the client system requests the original video segment from the cloud, essentially falling back to the normal VR rendering mode. We expect that the former case is common for many videos according to our study in § 5.1, thus significantly reducing client energy consumption.

5.3 Cloud Service Architecture

§ 5.1 confirms that users tend to track object movement. In addition, they tend to track not one object, but most often a group of objects.

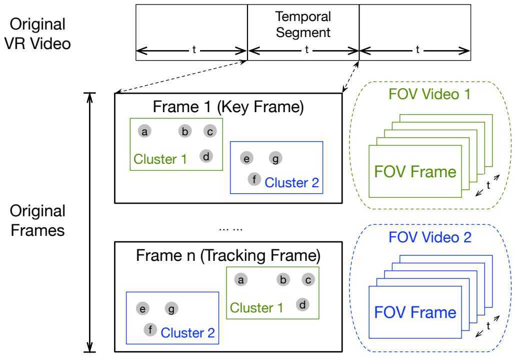


Figure 7: FOV video creation. a - g represent objects.

This motivates the fundamental idea behind the cloud architecture design: extract object information and group objects into different clusters. Each cluster contains a unique set of objects that users tend to watch together. By tracking the cluster of objects across frames, we can capture the general pattern of user's viewing areas.

FOV Video. We present the overview of how a VR video is processed in a cloud server in Figure 7. We first decompose the video into multiple t-second temporal segments. We categorize the frames of each video segment into two types: *key frame* and *tracking frame*. A key frame is always the first frame of a segment, it is a frame in which objects are explicitly detected and clustered. Objects within the same cluster are then tracked across subsequent tracking frames, effectively creating a trajectory of the object cluster.

For each cluster, the VR server creates a FOV video where each frame contains the object cluster and has a size that matches the end-user's FOV on the client device. The FOV frame is converted from the spherical format to the planar format that can be directly rendered on the VR device.

Temporal Segmentation. The size of the temporal segment determines the granularity of FOV videos. In one extreme design, one could create a FOV video for the entire video. This design would lead to high video compression rate [22], but is less flexible in that any single FOV-miss would lead to the re-streaming of the entire original video. Thus, we divide the original video into segments, each of which has its own FOV videos. In this way, only one segment of the original video needs to be transmitted upon a FOV-miss. We statically set the segment length to 30 frames, which roughly match the Group of Pictures (GOP) size in video compression [62].

SAS Store. The FOV videos is stored in the log-structured manner. We place the associated metadata in a separate log rather than mixing them with frame data. This allows us to decouple the metadata with video encoding, and thus simplifies the system design.

The server pre-renders the FOV videos statically, but this incurs storage overhead. An alternative would be to generate FOV videos on-demand when receiving user requests. We argue that sacrificing storage for lower latency is a desirable trade-off as the cloud storage becomes cheaper (less than $0.05 per GB for cloud storage [7]) while users demand higher viewing experience. The storage overhead is further amortized across all the end users. The exact overhead is determined by the number of FOV videos created, which in turn affects the FOV hit rate and thus the client-side energy savings. We will show in § 8.2 that SAS incurs modest storage overhead with significant energy savings.

Handling Client Requests. The SAS server differentiates between two types of client requests: requests for FOV videos and requests for the original video. The former is made at the beginning of each video segment when the client decides what object cluster the user is most likely interested in, and then the server delivers the corresponding FOV video. The latter is made when a FOV-miss happens, upon which the server serves the original segment.

5.4 Client Support for SAS

On the VR client, for each (FOV) frame that will be rendered, the playback application checks the real-time head pose and compares it against the associated metadata of the frame. If the desired FOV indicated by the current head pose is covered by the corresponding FOV frame (FOV-hit), the FOV frame can be directly rendered on the display. Otherwise (FOV-miss), the client will request the original video segment that contains the correct frame. It might initially seem to be wasteful to stream an entire segment although only the missing frames are needed. However, this strategy is more bandwidth-friendly because video compression rate is much higher than image compression rate [22]. As we will show in § 8.2, the buffering time to request a full segment is low and has insignificant impact on the overall user-experience.

6 Hardware Accelerated Rendering

SAS pre-renders VR content on the server to minimize the energy consumption on client devices. However, on-device rendering is still necessary either when users do not track objects or when the VR content does not go through a VR server (e.g., offline playback). Thus, it is critical to provide energy-efficient on-device rendering for EVR to be a general solution for VR video processing.

To this end, this section describes a hardware accelerator for efficient on-device rendering (HAR). The accelerator specifically targets the energy-intensive PT operations, which contribute to 40% of the compute and memory overhead of VR devices. We first provide an overview of the PT algorithm and explain why it is an ideal target for hardware acceleration (§ 6.1). We then describe the accelerator design (§ 6.2) and explain the design decisions (§ 6.3).

6.1 Algorithm Overview

Algorithmic Workflow. PT operation calculates the pixel values of the FOV frames (for display on the HMD) from the full frames in the input VR video. PT achieves so by mapping each FOV frame pixel to one pixel or a combination of different pixels in the corresponding input frame from the VR video. Formally, to calculate each pixel P (i, j) of the FOV frame, the PT algorithm goes through three processing stages: *perspective update*, *mapping*, and *filtering*. The perspective update stage calculates the (x, y, z) coordinates of the point P′ on the sphere that corresponds to P. The mapping stage then calculates the (u, v) coordinates of the point P′′ in the input frame that is projected from P′. Finally, the filtering stage uses (u, v) to sample the input frame to obtain the pixel value for P (i, j). PT iterates the same processing for all the pixels in the FOV frame.

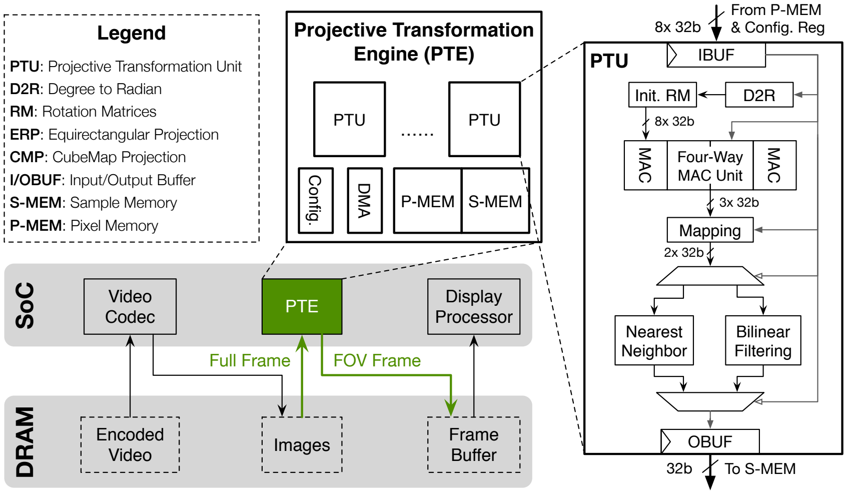


Figure 8: Overview of the augmented hardware architecture.

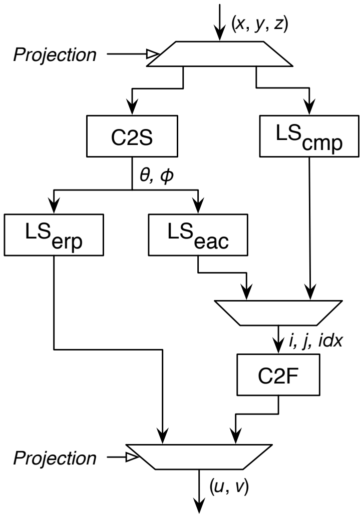


Figure 9: The mapping engine.

In today's VR client, PT is implemented as a *texture mapping* [38] task that is well-supported on modern (mobile) GPUs through the specialized Texture Mapping Units (TMU). Essentially, the input planar frame is treated as a texture that gets mapped to the sphere, on which the region that corresponds to user's current viewing area is projected to the two-dimensional space for display. This strategy eases development by reusing existing hardware (GPU) and software (OpenGL), but also introduces large overhead that constitutes about 40% of the VR processing energy (Figure 3b).

Specifically, GPUs provide hardware support to generic texture mapping that is unnecessary for PT. For instance, GPUs use dedicated cache memories to store texture data so as to efficiently support different data access patterns on the texture map [31]. However, input frames are accessed in a deterministic pattern in PT, and thus can be more efficiently managed by a scratchpad memory with lower energy. In addition, using the GPU for PT also necessarily invokes the entire software stack (application library, runtime, OS driver, etc.) that exacerbates the energy overhead.

Algorithmic Characteristics. PT algorithm exhibits three key characteristics that are amenable to hardware acceleration.

First, PT is massively parallel at the pixel level. Each pixel goes through the exact processing sequence, and the output is purely a function of the (i, j) coordinates of the pixel. This makes hardware parallelization and pipelining easier using classic techniques [48].

Second, the PT algorithm is compute-intensive. For each pixel, the algorithm takes in eight parameters such as the FOV size, HMD resolution, and head orientation, each of which is aligned to 32 bits, and returns a 24-bit RGB pixel value. The perspective update and mapping steps do not access frame pixel data, while the filtering step accesses only a few adjacent pixels, similar to a stencil operation. Overall, PT has high arithmetic intensity and strong data locality.

Finally, PT, just like texture mapping, is inherently inexact due to the filtering that reconstructs a pixel value from nearby pixels [66]. For this reason, most of the operations in the entire algorithm can be carried out in fixed-point arithmetics with little loss of user experience while making efficient use of hardware resources.

6.2 Hardware Architecture

We propose a new hardware accelerator, Projective Transformation Engine (PTE), that performs efficient projective transformations. We design the PTE as an SoC IP block that replaces the GPU and collaborates with other IPs such as the Video Codec and Display Processor for VR video rendering. Figure 8 shows how PTE fits into a complete VR hardware architecture. The PTE takes in frames that are decoded from the video codec, and produces FOV frames to the frame buffer for display. If a frame is already prepared by the cloud server as a projected FOV frame, the PTE sends it to the frame buffer directly; otherwise the input frame goes through the PTE's datapath to generate the FOV frame. The GPU can remain idle during VR video playback to save power.

The bulk of the PTE is a set of Projective Transformation Units (PTU) that exploit the pixel-level parallelism. The Pixel Memory (P-MEM) holds the pixel data for the incoming input frame, and the Sample Memory (S-MEM) holds the pixel data for the FOV frame that is to be sent to the frame buffer. The PTE uses DMA to transfer the input and FOV frame data. The PTE also provides a set of memory-mapped registers for configuration purposes. The configurability allows the PTE adapt to different popular projection methods and VR device parameters such as FOV size and display resolution. The configurability ensures PTE's flexibility without the overhead of general-purpose programmability that GPUs introduce.

Accelerator Memory. The P-MEM and S-MEM must be properly sized to minimize the DRAM traffic. Holding the entire input frame and FOV frame would require the P-MEM and S-MEM to match the video resolution (e.g., 4K) and display resolution (e.g., 1440p) respectively, requiring tens of MBs on-chip memories that are prohibitively large in practice. Interestingly, we find that the filtering step (the only step in the PT algorithm that access pixel data) operates much like a stencil operation that possess two properties.

First, PT accesses only a block of adjacent pixels for each input. Second, the accessed pixel blocks tend to overlap between adjacent inputs. Thus, the P-MEM and S-MEM are designed to hold several

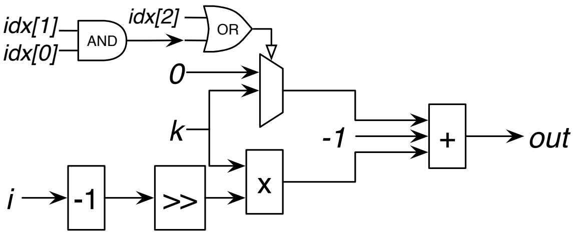


Figure 10: The mapping engine's *C2F* logic.

line of pixels in the input and FOV frame, which is similar to the *line-buffer* used in Image Signal Processor (ISP) designs [40].

PTU Microarchitecture. The PTU executes the PT algorithm for each pixel in the FOV frame to calculate its value. The microarchitecture of the PTU consists of three main parts that correspond to the three stages in the algorithm. The PTU is fully pipelined to accept a new pixel every cycle.

The *perspective update* module takes the current head orientation and, for each pixel P in the FOV frame, finds a point P′ on the sphere that corresponds to P. Computationally, this step is an affine transformation that multiplies the coordinate vector of P with two 3×3 rotation matrices preceded by a few pre-processing steps such as translating the angular form of the head orientation to cartesian coordinates. The rotation matrices are sparse by nature. In the hardware, the perspective update module is implemented by a four-way fixed-point MAC unit that uniquely suits the sparsity.

The *mapping* module maps the spherical point P′ to a point P′′ in the input frame, essentially projecting P′ to P′′ according to the particular projection method in which the VR video is originally created (Figure 1). Our current design supports three commonly-used projection methods: Equirectangular Projection (ERP), CubeMap Projection (CMP) [18], and Equi-Angular Cubemap (EAC) [3]. The former directly maps a point on a sphere to a rectangular frame according to its latitude and longitude; the latter two deform a sphere into a cube, whose six faces get unfolded and laid flat. The three projection methods have different trade-offs that are beyond the scope of this paper [27], but are widely used in different scenarios, and thus must be efficiently supported in the PTU.

Our key architectural observation is that the three projection methods share similar building blocks, which exposes an opportunity for a modular hardware design that can be easily configured to support any given method. Equ. 1 - Equ. 3 show the computation structures of the three methods and their modularities. Specifically, both ERP and EAC require Cartesian-to-Spherical transformation (*C2S*); both EAC and CMP require Cube-to-Frame transformation (*C2F*); they all require a different linear scaling (*LS*). Figure 9 illustrates the hardware design of the mapping hardware where *C2S* and *C2F* are reused across projection methods.

$$ERP: \; C2S \circ LS_{erp} \tag{1}$$

$$EAC: \; C2S \circ LS_{eac} \circ C2F \tag{2}$$

$$CMP: \; LS_{cmp} \circ C2F \tag{3}$$

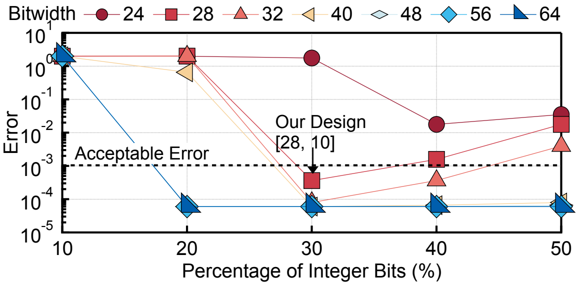


Figure 11: The pixel error rate of the FOV frame changes with fixed-point representations.

The computation of the mapping engine is simple, especially when using a fixed-point implementation. To illustrate its complexity, Figure 10 shows the *C2F* logic in the mapping engine. The critical path is dominated by the multipler.

The *filtering* module indexes into the input frame using takes the coordinates of P′′, and assigns the returned pixel value to P. If P′′ happens to map to an integer pixel in the input frame, the pixel value at the location of P′′ can simply be assigned to P. Otherwise, the hardware will reconstruct the pixel value using pixels adjacent to P′′ by applying a so called *filtering function*. Our PTU design supports two classic filtering functions: nearest neighbor and bilinear interpolation.

6.3 Design Decisions

SoC Integration. We design the PTE as a standalone IP block in order to enable modularity and ease distribution. Alternatively, we envision that the PTE logic could be tightly integrated into either the Video Codec or Display Processor. Indeed, many new designs of the Display Processor have started integrating functionalities that used to be executed in GPUs such as color space conversion [2]. Such a tight integration would let the Display Processor directly perform PT operations before scanning out the frame to the display, and thus reduces the memory traffic induced by writing the FOV frames from the PTE to the frame buffer. However, our principal idea of bypassing the GPU as well as the PTE microarchitecture are still fundamental to such a design.

Optimization Choices. Our design goal is to reduce the energy consumption rather than improving the frame rate (i.e., throughput) as today's VR devices are mostly able to render VR videos in real-time (30 FPS). We thus do not pursue architectural optimizations that improve throughput beyond real-time at the cost of energy overhead. For instance, the perspective update module in the PTU does not batch the vector-matrix multiplications of different pixels as matrix-matrix multiplications because the latter improves performance at the cost of more hardware resources and energy.

Fixed-Point Computation. We also quantitatively determine the bitwidth used for the fixed-point computations in PTE. Figure 11 shows how the average pixel error changes with the total bitwidth and the percentage used for the integer part. We confirm that an average pixel error below 10^{-3} is visually indifferentiable. Thus, we choose a 28-bit representation with 10 bits for the integer part (denoted as [28, 10] in Figure 11). Other designs either waste energy or exceed the error threshold.

7 Implementation

Our prototypical implementation of EVR is distributed across a VR content server (§ 7.1) and a playback client (§ 7.2).

7.1 Cloud Server Implementation

The server is hosted on an Amazon EC2 t2.micro instance with the videos, including the original VR videos and FOV videos, stored on an S3 instance in the Standard Storage class.

We use a convolutional neural network, YOLOv2 [60], for object detection for its superior accuracy in the VR server. The server uses the classic k-means algorithm [34] for object clustering based on the intuition that users tend to watch objects that are close to each other. Future explorations could exploit clustering techniques that leverage even richer object semantics such as object category.

7.2 Client Implementation

The EVR client has two responsibilities: interacting with the server to participate in semantic-aware streaming and performing hardware-accelerated rendering. The client is implemented on a customize platform to emulate a hypothetical VR device augmented with the PTE accelerator. The platform combines an NVidia Jetson TX2 development board [9] with a Xilinx Zynq-7000 SoC ZC706 board [20].

The TX2 board contains a state-of-the-art Tegra X2 mobile SoC [13]. TX2 is used in contemporary VR systems, including ones from Magic Leap [10] and GameFace [14]. In addition, many VR devices such as Samsung Gear VR and Oculus Go use smartphone-grade SoCs such as Qualcomm Snapdragon 821, which have the capabilities similar to TX2. TX2 also allows us to conduct component-wise power measurement, which is not obtainable from off-the-shelf VR devices. The client player leverages TX2’s hardware-accelerated Video Codec through the GStreamer framework [6, 57].

The Zynq board let us prototype the PTE accelerator that would not be feasible on TX2 alone. We do not prototype the whole system on the Zynq board because it lacks efficient CPU/GPU/video Codec that a typical VR device possesses. We implement the PTE accelerator in RTL, and layout the design targeting the 28 nm FPGA fabric on the Xilinx Zynq-7000 SoC ZC706 board. The available resources allows us to instantiate 2 PTUs with P-MEM and S-MEM sized at 512 KB and 256 KB, respectively.

Post-layout results show that the PTE accelerator can operate at 100 MHz and consumes 194 mW of power, indicating one order of magnitude power reduction compared to a typical mobile GPU. The PTU is fully pipelined to accept a new pixel every cycle. Operating at 100 MHz, the PTE delivers 50 FPS, sufficient for real-time VR. The performance and power results reported should be seen as lower-bounds as an ASIC flow would yield better energy-efficiency.

8 Evaluation

We first introduce the evaluation methodology (§ 8.1). We then evaluate EVR over three key VR use-cases: online streaming (§ 8.2), live streaming (§ 8.3), and offline playback (§ 8.4). We then show that EVR out-performs an alternative design that directly predicts head motion on-device (§ 8.5). Finally, we show the general applicability of the PTE hardware beyond 360° video rendering (§ 8.6).

8.1 Evaluation Methodology

Usage Scenarios. We evaluate three EVR variants, each applies to a different use-case, to demonstrate EVR’s effectiveness and general applicability. The three variants are:

- S: leverages SAS without HAR.
- H: uses HAR without SAS.
- S+H: combines the two techniques.

The three settings are evaluated under three VR use-cases:

- *Online-Streaming*: The VR content is streamed from a VR server and played back on the VR client device. All three settings above apply to this use-case.
- *Live-Streaming*: The VR content is streamed from a capture device to the VR client device (e.g., broadcasting a sports event). Although VR videos still go through a content server (e.g., YouTube) in live-streaming, the server does not perform sophisticated processing due to the real-time constraints [54]. Therefore, SAS is not available, and only the setting H is applicable.
- *Offline-Playback*: The VR content is played back from the local storage on the VR client. Only the setting H applies.

Energy Evaluation Framework. Our energy evaluation framework considers the five important components of a VR device: network, display, storage, memory, and compute. The network, memory, and compute power can be directly measured from the TX2 board through the on-board Texas Instruments INA 3221 voltage monitor IC. We also use a 2560 × 1440 AMOLED display that is used in Samsung Gear VR and its power is measured in our evaluation. We estimate the storage energy using an empirical eMMC energy model [41] driven by the storage traffic traces.

The total energy consumption of the five components is reported for S. To evaluate the energy consumption of H and S+H, we replace the GPU power consumed during projection transformation with the post-layout FPGA power.

Baseline. We compare against a baseline that is implemented on the TX2 board and that does not use SAS and HAR. The baseline is able to deliver a real-time (30 FPS basis) user-experience. Our goal is to show that EVR can effectively reduce the energy consumption with little loss of user-experience.

Benchmark. To faithfully represent real VR user behaviors, we use a recently published VR video dataset [25], which consists of head movement traces from 59 real users viewing different 360° VR videos on YouTube. The videos have a 4K (3840 × 2160) resolution, which is regarded as providing an immersive VR experience. The dataset is collected using the Razer Open Source Virtual Reality (OSVR) HDK2 HMD with an FOV of 110° × 110° [16], and records users’ real-time head movement traces. We replay the traces to emulate readings from the IMU sensor and thereby mimic realistic VR viewing behaviors. This trace-driven methodology ensures the reproducibility of our results.

8.2 Online-Streaming Use-case Evaluation

Energy Reductions. We quantify the energy saving of the three EVR variants over the baseline in Figure 12. The left y-axis shows the compute (SoC) energy savings and the right y-axis shows the device-level energy savings.

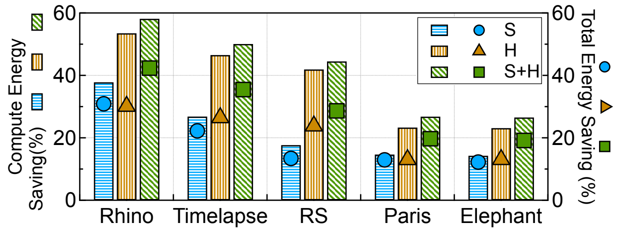


Figure 12: Normalized energy consumption across different EVR variants. S+H delivers the highest energy savings.

On average, S and H achieve 22% and 38% compute energy savings. S+H combines SAS and HAR and delivers an average 41%, and up to 58%, energy saving. The compute energy savings across applications are directly proportional to the PT operation's contributions to the processing energy as shown in Figure 3b. For instance, Paris and Elephant have lower energy savings because their PT operations contribute less to the total compute energy consumptions.

The trend is similar for the total device energy savings. S+H achieves on average 29% and up to 42% energy reduction. The energy reduction increases the VR viewing time, and also reduces the heat dissipation and thus provides a better viewing experience.

User Experience Impact. We quantify user experience both quantitatively and qualitatively. Quantitatively, we evaluate the percentage of FPS degradation introduced by EVR compared to the baseline. Figure 13 shows that the FPS drop rate averaged across 59 users is only about 1%. Lee et al., reported that a 5% FPS drop is unlikely to affect user perception [47]. We assessed qualitative user experience and confirmed that the FPS drop is visually indistinguishable and that EVR delivers smooth user experiences.

The FPS drops come from FOV misses introduced by SAS. Our profiling shows that SAS introduces an average FOV-miss rate of 7.7% when streaming the VR videos used in our evaluation. Specifically, the FOV-miss rate ranges from 5.3% for Timelapse to 12.0% for RS. Under the WiFi environment (with an effective bandwidth of 300 Mbps) where our experiments are conducted, every re-buffering of a missed segment pauses rendering for at most 8 milliseconds.

Bandwidth Savings. Although the goal of EVR is not to save bandwidth, EVR does reduce the network bandwidth requirement through SAS, which transmits only the pixels that fall within user's sight. The right y-axis of Figure 13 quantifies the bandwidth saving of S+H compared to the baseline system that always streams full frames. EVR reduces the bandwidth requirement by up to 34% and 28% on average. We expect that combining head movement prediction [36, 58] with SAS would further improve the bandwidth efficiency, which we wish to develop as future work.

Storage Overhead. EVR introduces storage overhead by storing FOV videos. The exact storage overhead depends on the "object utilization", which denotes the percentage of objects used for creating FOV videos, which in turn affects energy savings. Using more objects to create FOV videos leads to more FOV hits and thus more energy savings, but also incurs higher storage overhead as more FOV videos must be stored.

We quantify the storage-energy trade-off by varying the object utilization from 25%, 50%, 75%, to 100%. Figure 14 illustrates the results where the x-axis shows the storage overhead normalized to the original VR video sizes under the four utilizations, and the y-axis shows the energy savings of S+H. At an 100% object utilization, the average storage overhead is 4.2×, Paris and Timelapse have the lowest and highest overhead of 2.0× and 7.6×, respectively. We note that the storage overhead incurs little extra monetary cost for streaming service providers given that the cloud storage has become as cheap as $0.05 per GB [7]. Specifically, the extra storage incurs on average $0.02 cost per video. This cost will be further amortized across millions of users that stream the video, and is negligible compared to the over $150 custom acquisition cost that video companies already pay [12].

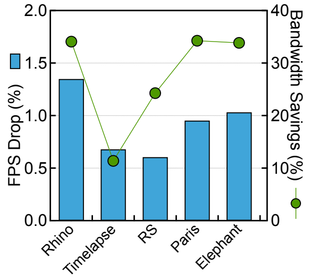


Figure 13: FPS drop and bandwidth reduction.

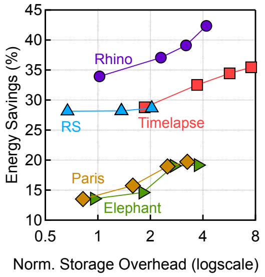


Figure 14: Storage overhead and energy saving trade-off.

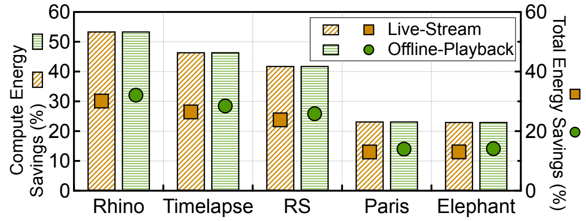


Figure 15: Compute and total energy savings of H in the offline-playback and live-stream use-cases.

As the object utilization decreases to 25%, the storage overhead and the energy savings also decrease. At a 25% object utilization, EVR incurs an average storage overhead of only 1.1×, but still delivers an average 24% energy saving. This shows that the object utilization is a powerful knob for storage-energy trade-off.

8.3 Live-Streaming Use-case Evaluation

We now evaluate EVR on the live-stream scenario to mimic live broadcasting, in which only H applies. Figure 15 shows the energy saving of H over the baseline. Using hardware-accelerated rendering, H achieves 38% compute energy savings (left y-axis) and 21% device energy savings (right y-axis). Comparing against the online-streaming use-case, live-streaming has lower energy savings because live-streaming can not take advantage of semantic-aware streaming due to its real-time requirements.

8.4 Offline-Playback Use-case Evaluation

We also evaluate the efficiency of the hardware-accelerated rendering with offline-playback cases. We show the energy saving of H over the baseline in Figure 15. The compute energy saving (left y-axis) is similar to the live-stream, but the device energy

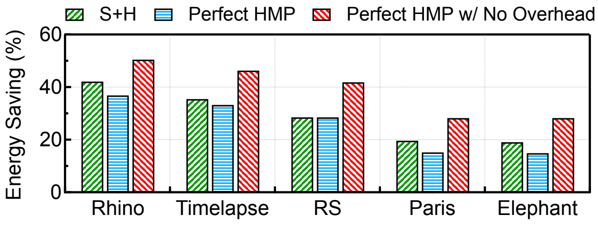


Figure 16: Energy savings of S+H compared against the scheme that uses on-device head motion prediction (HMP).

saving (right y-axis) is slightly higher (23% vs. 21%) because offline-playback does not consume network energy, and thus compute energy saving contributes more to the total device energy saving.

8.5 SAS vs. Client Head Motion Prediction

The SAS uses object semantics to track user viewing areas without requiring client support, which reduces the energy consumption of the client device. An alternative would be to predict head motion directly on the client device. This does not require cloud servers to track object trajectories, but it could incur computation overhead on the client device due to the running of prediction models.

To compare with this alternative, we integrate a recently proposed deep neural network (DNN)-based head movement predictor (HMP) [56] into SAS. Since the original DNN is not trained on our dataset, we assume that the prediction network has a *perfect (100%)* prediction accuracy. We also generously assume that the server pre-renders all the FOV videos that correspond to all the possible head orientations. Thus, the FOV videos can be directly streamed and rendered without PT operations. To ensure a low compute overhead for the DNN prediction, we assume that the client device's SoC employs a dedicated DNN accelerator. We model the accelerator using a cycle-accurate DNN simulator SCALESim [64]. We assume a 24 × 24 systolic array operating at 1 GHz to represent a typical mobile DNN accelerator [72].

We show the device-level energy comparison in Figure 16. Our EVR design saves more energy than the system with a perfect on-device head motion prediction (29% vs. 26%). This is because the predictor introduces high on-device energy overhead. That said, we believe the HMP overhead will decrease with both algorithmic and architectural innovations, and EVR can readily leverage it. We build an ideal EVR, in which SAS uses a HMP that has a perfect prediction with *no* overhead. As shown in Figure 16, EVR with this ideal predictor can improve the average energy saving to 39%.

8.6 General Applicability of PTE Hardware

Fundamentally, the proposed PTE hardware is specialized for PT operations, which is critical to all use-cases that involve panoramic content. 360° video rendering is just one of them. To demonstrate the general applicability of PTE, we evaluate another use-case: 360° video quality assessment on content servers. This use-case quantifies the visual quality of 360° content in real-time to avoid processing low-quality content. The quality assessor first performs a sequence of PT operations to project the content to viewers' perspective, and calculates metrics such as Peak Signal to Noise Ratio and Structural Similarity Index to assess the video quality.

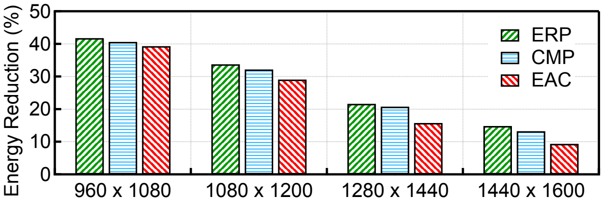


Figure 17: Energy reductions of using PTE over a GPU-based baseline for real-time 360° video quality assessment.

We compare the energy consumption of 360° video quality assessment between an optimized GPU-based system and a PTE-augmented system. The GPU baseline is implemented according to a recently proposed quality assessment pipeline [68]. The results are shown in Figure 17. We vary the output resolution to show the sensitivity. We find that the PTE achieves up to 40% energy reduction. The reduction decreases as the resolution increases because the GPU better amortizes the cost over more pixel processings.

9 Related Work

VR Energy Optimizations. While there exists abundant prior work on optimizing the energy consumption of smartphones [24, 42, 67, 71] and werables [33, 41, 50, 52, 65], VR devices receive little attention. Our characterizations show a different power profile from conventional mobile devices. Specifically, display, network, and storage consume little power. Thus, EVR specifically targets reducing the compute energy rather than the other components [45].

Remote-rendering has been exploited in the context of gaming [47, 51] by dynamically offloading the rendering tasks to a remote server and applying optimizations to hide the network latency. In contrast, EVR *pre-renders* VR content statically without dynamic offloading. Boos et al. [23] proposes to pre-render all possible images in a VR game to save energy. In contrast, EVR leverages the inherent object semantics to *selectively* pre-render only part of VR videos. LightDB [35] introduces a VR video database management system based on a new representation of VR content. LightDB accelerates various DB queries on the VR videos while EVR focuses on reducing energy consumption of video rendering.

VR Bandwidth Optimizations. The vast majority of today's VR systems focus on optimizing network bandwidth requirement. Most notably, researchers have started taking into account user viewing area and investigating view-guided optimizations [26, 27, 29, 37]. They share the same key idea: divide a frame into tiles and use non-uniform image resolutions across tiles according to users' sight. For instance, Haynes et al. [36] and Zare et al. [70] both propose to predict user head movement and thereby reducing the resolution of out-of-sight tiles. Similarly, Khiem et al. [44, 59] proposes to predict users' Region-of-Interest (ROI) and streams the ROIs with high resolution. Qian et al. [58], Fan et al. [28], and Liu et al. [53] propose to stream only the user's viewing tiles.

The SAS part of EVR can also be categorized as an view-guided approach. However, SAS is fundamentally an *energy optimization* rather than a bandwidth optimization, and thus offers energy advantages. Specifically, most of existing view-guided streaming schemes require transferring the whole area of the frames. Thus, the power-hungry PT operation is still a necessary step on the VR device. In contrast, EVR reduces the device energy by providing specialized hardware for the PT operation and/or pre-executing the PT step in the cloud. Leng et al. [49] presents semantics-aware streaming while our work integrates SAS with HAR.

VR Capturing. VR content is created by special capturing devices such as omnidirectional cameras [21, 43] or multi-camera rigs [4]. Google [5], Facebook [4], and Samsung [17] have all released hardware and software to streamline the VR video capturing process. Konrad et al. proposes a system that natively generates spherical VR videos while bypassing most of the compute-intensive processing in conventional VR capture systems [46]. Mazumdar et al. proposes specialized hardware to accelerate the compute-intensive bilateral solver during VR capturing [55]. They also find that using carefully-tuned fixed-point implementations greatly reduces resource utilization with little loss of quality.

Orthogonal to the capturing systems, EVR instead focuses on VR playback because it significantly impacts the VR device energy. Thus far, VR content capture and playback are considered as two separate processes and are optimized in isolation. To the best of our knowledge, none of the existing off-the-shelf VR playback systems uses customized hardware for 360° video streaming. We believe that co-designing capture and playback systems might bring even greater benefits. For instance, it would be interesting to study how the capturing system can encode semantics information in the VR content and thus simplify the design of the VR playback systems. We would like to explore this as the future work.

10 Conclusion

The rapid growth of 360° content is fueling the adoption of VR. Soon 360° content will soon be consumed just any other media. However, processing 360° content introduces significant energy-inefficiencies to today's mobile systems. The fundamental reason is that today's mobile SoCs are tuned for conventional planar video processing. We re-examine video processing in light of this paradigm shift, and discover that the key toward energy-efficient VR video processing is to collaboratively co-optimize the cloud server with the device hardware architecture. We hope our work serves the first step in a promising new direction of research.

Acknowledgments

We would like to thank the anonymous reviewers for their helpful comments and feedback.

References

[1] Amazon Silk. https://docs.aws.amazon.com/silk/latest/developerguide/introduction.html.

[2] ARM Mali Display Processors. https://developer.arm.com/products/graphics-and-multimedia/mali-display-processors.

[3] Bringing pixels front and center in vr video. https://blog.google/products/google-vr/bringing-pixels-front-and-center-vr-video/.

[4] Facebook Surround 360. https://facebook360.fb.com/facebook-surround-360/.

[5] Google Jump VR. https://vr.google.com/jump/.

[6] GStreamer. https://gstreamer.freedesktop.org/.

[7] Hard Drive Cost Per Gigabyte. https://www.backblaze.com/blog/hard-drive-cost-per-gigabyte/.

[8] Hospital-wide access to virtual reality alleviates pain and anxiety for pediatric patients. http://www.stanfordchildrens.org/en/about/news/releases/2017/virtual-reality-alleviates-pain-anxiety.

[9] Jetson TX2 Module. http://www.nvidia.com/object/embedded-systems-dev-kits-modules.html.

[10] Magic Leap One Powered by Nvidia Tegra TX2. https://www.tomshardware.com/news/magic-leap-tegra-specs-release,37443.html.

[11] N16886, Joint Call for Evidence on Video Compression with Capability beyond HEVC. https://mpeg.chiariglione.org/standards/exploration/future-video-coding/n16886-joint-call-evidence-video-compression-capability.

[12] Netflix's 8.3 Million New Subscribers Didn't Come Cheap. https://www.fool.com/investing/2018/01/23/netflixs-83-million-new-subscribers-didnt-come-che.aspx.

[13] Nvidia's Jetson TX2 Powers GameFace Labs' Standalone VR Headset. https://gstreamer.freedesktop.org://www.tomshardware.com/news/gameface-labs-standalone-steamvr-headset,37112.html.

[14] Nvidia's Jetson TX2 Powers GameFace Labs' Standalone VR Headset. https://www.tomshardware.com/news/gameface-labs-standalone-steamvr-headset,37112.html.

[15] Opioids haven't solved chronic pain. maybe virtual reality can. https://www.wired.com/story/opioids-havent-solved-chronic-pain-maybe-virtual-reality-can/.

[16] Razer OSVR HDK2. https://versus.com/en/htc-vive-vs-razer-osvr-hdk2.

[17] Samsung Project Beyond. http://thinktankteam.info/beyond/.

[18] Under the hood: Building 360 video. https://code.facebook.com/posts/1638767863078802.

[19] White Paper: 360-Degree Video Rendering. https://community.arm.com/graphics/b/blog/posts/white-paper-360-degree-video-rendering.

[20] Xilinx Zynq-7000 SoC ZC702 Evaluation Kit. https://www.xilinx.com/products/boards-and-kits/ek-z7-zc702-g.html.

[21] Robert Anderson, David Gallup, Jonathan T Barron, Janne Kontkanen, Noah Snavely, Carlos Hernández, Sameer Agarwal, and Steven M Seitz. Jump: virtual reality video. *ACM Transactions on Graphics (TOG)*, 35(6):198, 2016.

[22] AXIS Communications. *An explanation of video compression techniques.* White Paper.

[23] Kevin Boos, David Chu, and Eduardo Cuervo. Flashback: Immersive virtual reality on mobile devices via rendering memoization. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys'16)*, pages 291–304. ACM, 2016.

[24] Xiaomeng Chen, Ning Ding, Abhilash Jindal, Y Charlie Hu, Maruti Gupta, and Rath Vannithamby. Smartphone energy drain in the wild: Analysis and implications. *ACM SIGMETRICS Performance Evaluation Review*, 43(1):151–164, 2015.

[25] Xavier Corbillon, Francesca De Simone, and Gwendal Simon. 360-degree video head movement dataset. In *Proceedings of the 8th ACM on Multimedia Systems Conference*, pages 199–204. ACM, 2017.

[26] Lucia D'Acunto, Jorrit van den Berg, Emmanuel Thomas, and Omar Niamut. Using mpeg dash srd for zoomable and navigable video. In *Proceedings of the 7th International Conference on Multimedia Systems*, page 34. ACM, 2016.

[27] Tarek El-Ganainy and Mohamed Hefeeda. Streaming virtual reality content. *arXiv preprint arXiv:1612.08350*, 2016.

[28] 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 *Proceedings of the 27th Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 67–72. ACM, 2017.

[29] Vamsidhar Reddy Gaddam, Michael Riegler, Ragnhild Eg, Carsten Griwodz, and Pål Halvorsen. Tiling in interactive panoramic video: Approaches and evaluation. *IEEE Transactions on Multimedia*, 18(9):1819–1831, 2016.

[30] Mario Graf, Christian Timmerer, and Christopher Mueller. Towards bandwidth efficient adaptive streaming of omnidirectional video over http: Design, implementation, and evaluation. In *Proceedings of the 8th ACM on Multimedia Systems Conference*, pages 261–271. ACM, 2017.

[31] Ziyad S Hakura and Anoop Gupta. The design and analysis of a cache architecture for texture mapping. *ACM SIGARCH Computer Architecture News*, 25(2):108–120, 1997.

[32] Matthew Halpern, Yuhao Zhu, and Vijay Janapa Reddi. Mobile CPU's Rise to Power: Quantifying the Impact of Generational Mobile CPU Design Trends on

Performance, Energy, and User Satisfaction. In *Proc. of HPCA*, 2016.

[33] MyungJoo Ham, Inki Dae, and Chanwoo Choi. Lpd: Low power display mechanism for mobile and wearable devices. In *USENIX Annual Technical Conference*, 2015.

[34] John A Hartigan and Manchek A Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979.

[35] Brandon Haynes, Amrita Mazumdar, Armin Alaghi, Magdalena Balazinska, Luis Ceze, and Alvin Cheung. Lightdb: A dbms for virtual reality video. *Proceedings of the VLDB Endowment*, 11(10), 2018.

[36] Brandon Haynes, Artem Minyaylov, Magdalena Balazinska, Luis Ceze, and Alvin Cheung. Visualcloud demonstration: A dbms for virtual reality. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 1615–1618. ACM, 2017.

[37] Jian He, Mubashir Adnan Qureshi, Lili Qiu, Jin Li, Feng Li, and Lei Han. Rubiks: Practical 360-degree streaming for smartphones. In *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys'18)*, 2018.

[38] Paul S Heckbert. Survey of texture mapping. *IEEE computer graphics and applications*, 6(11):56–67, 1986.

[39] Paul S Heckbert. Fundamentals of texture mapping and image warping. master's thesis. *University of California, Berkeley*, 2:3, 1989.

[40] James Hegarty, John Brunhaver, Zachary DeVito, Jonathan Ragan-Kelley, Noy Cohen, Steven Bell, Artem Vasilyev, Mark Horowitz, and Pat Hanrahan. Darkroom: Compiling High-Level Image Processing Code into Hardware Pipelines. In *Proc. of SIGGRAPH*, 2014.

[41] Jian Huang, Anirudh Badam, Ranveer Chandra, and Edmund B Nightingale. Weardrive: Fast and energy-efficient storage for wearables. In *USENIX Annual Technical Conference*, pages 613–625, 2015.

[42] Junxian Huang, Feng Qian, Alexandre Gerber, Z Morley Mao, Subhabrata Sen, and Oliver Spatscheck. A close examination of performance and power characteristics of 4g lte networks. In *Proceedings of the 10th international conference on Mobile systems, applications, and services*, 2012.

[43] Hiroshi Ishiguro, Masashi Yamamoto, and Saburo Tsuji. Omni-directional stereo. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):257–262, 1992.

[44] Ngo Quang Minh Khiem, Guntur Ravindra, and Wei Tsang Ooi. Adaptive encoding of zoomable video streams based on user access pattern. *Signal Processing: Image Communication*, 27(4):360–377, 2012.

[45] Tan Kiat Wee, Eduardo Cuervo, and Rajesh Krishna Balan. Demo: Focusvr: Effective & usable vr display power management. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services Companion*, pages 122–122. ACM, 2016.

[46] Robert Konrad, Donald G Dansereau, Aniq Masood, and Gordon Wetzstein. Spinvr: towards live-streaming 3d virtual reality video. *ACM Transactions on Graphics (TOG)*, 36(6):209, 2017.

[47] Kyungmin Lee, David Chu, Eduardo Cuervo, Johannes Kopf, Yury Degtyarev, Sergey Grizan, Alec Wolman, and Jason Flinn. Outatime: Using speculation to enable low-latency continuous interaction for mobile cloud gaming. In *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, pages 151–165. ACM, 2015.

[48] Charles E Leiserson and James B Saxe. Retiming synchronous circuitry. *Algorithmica*, 6(1-6):5–35, 1991.

[49] Yue Leng, Chi-Chun Chen, Qiuyue Sun, Jian Huang, and Yuhao Zhu. Semantic-aware virtual reality video streaming. In *Proceedings of the 9th Asia-Pacific Workshop on Systems*, page 21. ACM, 2018.

[50] Jing Li, Anirudh Badam, Ranveer Chandra, Steven Swanson, Bruce L Worthington, and Qi Zhang. On the energy overhead of mobile storage systems. In *FAST*, pages 105–118, 2014.

[51] Luyang Liu, Ruiguang Zhong, Wuyang Zhang, Yunxin Liu, Jiansong Zhang, Lintao Zhang, and Marco Gruteser. Cutting the cord: Designing a high-quality untethered vr system with low latency remote rendering. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys'18)*, 2018.

[52] Xing Liu and Feng Qian. Poster: Measuring and optimizing android smartwatch energy consumption: poster. In *Proceedings of the 22nd Annual International Conference on Mobile Computing and Networking*, pages 421–423. ACM, 2016.

[53] Xing Liu, Qingyang Xiao, Vijay Gopalakrishnan, Bo Han, Feng Qian, and Matteo Varvello. 360 innovations for panoramic video streaming. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*, pages 50–56. ACM, 2017.

[54] Andrea Lottarini, Alex Ramirez, Joel Coburn, Martha A Kim, Parthasarathy Ranganathan, Daniel Stodolsky, and Mark Wachsler. vbench: Benchmarking video transcoding in the cloud. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 797–809. ACM, 2018.

[55] Amrita Mazumdar, Armin Alaghi, Jonathan T Barron, David Gallup, Luis Ceze, Mark Oskin, and Steven M Seitz. A hardware-friendly bilateral solver for real-time virtual reality video. In *Proceedings of High Performance Graphics*, page 13. ACM, 2017.

[56] 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 *2018 ACM Multimedia Conference on Multimedia Conference*, pages 1190–1198. ACM, 2018.

[57] Nvidia. *Accelerated GStreamer User Guide, Release 28.2.*

[58] Feng Qian, Lusheng Ji, Bo Han, and Vijay Gopalakrishnan. Optimizing 360 video delivery over cellular networks. In *Proceedings of the 5th Workshop on All Things Cellular: Operations, Applications and Challenges*, pages 1–6. ACM, 2016.

[59] Ngo Quang Minh Khiem, Guntur Ravindra, Axel Carlier, and Wei Tsang Ooi. Supporting zoomable video streams with dynamic region-of-interest cropping. In *Proceedings of the first annual ACM SIGMM conference on Multimedia systems*, pages 259–270. ACM, 2010.

[60] Joseph Redmon and Ali Farhadi. YOLO9000: Better, Faster, Stronger. In *arXiv:1612.08242*, 2016.

[61] ABI Research. Augmented and Virtual Reality: The First Wave of 5G Killer Apps. 2017.

[62] Iain E Richardson. *The H. 264 advanced video compression standard*. John Wiley & Sons, 2011.

[63] Patrice Rondao Alface, Jean-François Macq, and Nico Verzijp. Interactive omnidirectional video delivery: A bandwidth-effective approach. *Bell Labs Technical Journal*, 16(4):135–147, 2012.

[64] Ananda Samajdar, Yuhao Zhu, Paul Whatmough, Matthew Mattina, and Tushar Krishna. Scale-sim: Systolic cnn accelerator. *arXiv preprint arXiv:1811.02883*, 2018.

[65] G Enrico Santagati and Tommaso Melodia. U-wear: Software-defined ultrasonic networking for wearable devices. In *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, pages 241–256. ACM, 2015.

[66] Dave Shreiner, Bill The Khronos OpenGL ARB Working Group, et al. *OpenGL programming guide: the official guide to learning OpenGL, versions 3.0 and 3.1*. Pearson Education, 2009.

[67] Narendran Thiagarajan, Gaurav Aggarwal, Angela Nicoara, Dan Boneh, and Jatinder Pal Singh. Who killed my battery?: analyzing mobile browser energy consumption. In *Proceedings of the 21st international conference on World Wide Web*, pages 41–50. ACM, 2012.

[68] Huyen TT Tran, Cuong T Pham, Nam Pham Ngoc, Anh T Pham, and Truong Cong Thang. A study on quality metrics for 360 video communications. *IEICE TRANSACTIONS on Information and Systems*, 101(1):28–36, 2018.

[69] Ville Ukonaho. Global 360 camera sales forecast by segment: 2016 to 2022. 2017.

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

[71] Yuhao Zhu and Vijay Janapa Reddi. Webcore: Architectural support for mobile web browsing. In *Proceeding of the International Symposium on Computer Architecuture*, pages 541–552, 2014.

[72] Yuhao Zhu, Anand Samajdar, Matthew Mattina, and Paul Whatmough. Euphrates: Algorithm-soc co-design for energy-efficient mobile continuous vision. *Proc. of ISCA*, 2018.