OPTIMIZING EMERGING MULTIMEDIA SYSTEMS:

A PRINCIPLED APPROACH TO ENHANCING QUALITY-OF-EXPERIENCE

by

Anlan Zhang

A Dissertation Presented to the
FACULTY OF THE USC GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
(COMPUTER ENGINEERING)

May 2025

Dedication

Dedicated to my beloved parents, Xinghua Xu and Longman Zhang.

In loving memory of my grandmother, Chunxiang Du, and my grandfather, Rongren Xu.

Acknowledgements

Much like rowing a boat across a vast ocean, the six-year journey of my Ph.D. study has been one of the most challenging yet rewarding experiences of my life. Completing this dissertation would not have been possible without the support, guidance, and encouragement of many individuals.

I would like to express my deepest gratitude to my Ph.D. advisor, Professor Feng Qian – the lighthouse guiding my boat to shore. As an expert in mobile computing, networking, and systems, Feng provided insightful and constructive feedback on my research projects and consistently helped me navigate the right path whenever I lost my way. Beyond that, Feng helped me grow from a novice undergrad to an independent researcher. Under his guidance, I learned how to conduct research professionally – from formulating research questions and solving complex problems to summarizing and communicating findings effectively. More importantly, Feng's diligence, passion, and patience have set a role model that I aspire to follow in both my academic and personal development. It was my great pleasure to work with him since August 2019.

I am deeply thankful to Professor Bo Han, the North Star throughout my doctoral journey. I have worked closely with Bo since August 2019. His exceptional guidance has greatly helped with both the completion of my dissertation and my academic development.

I am profoundly grateful to Professor C.-C. Jay Kuo and Professor Harsha V. Madhyastha for serving on my dissertation committee. Their valuable comments and suggestions were instrumental in improving my dissertation, and I sincerely appreciate their presence at my oral defense. I would also like to thank Professor Ramesh Govindan and Professor Murali Annavaram for their constructive feedback on my thesis proposal.

Over the past six years, I have had multiple internships in the industry and have greatly benefited from the guidance of my mentors and the collaboration with my colleagues. I was fortunate to work with knowledgeable and experienced researchers and engineers: Stefano Petrangeli, Haoliang Wang, Yu Shen, Saayan Mitra, Vishy Swaminathan, Chia-Yang Tsai, Ahmed Fouad, *etc.* Additionally, I have had the pleasure of collaborating with exceptional researchers in academia, including Professor Zhi-Li Zhang and Professor Z. Morley Mao, whose valuable insights have significantly contributed to my academic development. I truly enjoyed working with them.

My heartfelt thanks also go to my friends and fellow colleagues, including but not limited to: Chendong Wang, Kate Li Eidem, Haoyu Gong, Zhaokang Ke, Huibing Dong, Wenlong Wang, Yuanhao Liu, Cheng Chang, Zejun Zhang, Xing Liu, Bo Yan, Yu Liu, Yuming Hu, Wei Ye, Bin Hu, Ahmad Hassan, Xumiao Zhang, Ruiyang Zhu, Xiao Zhu, Shichang Xu, and many others. I am incredibly fortunate to have them with me along the way.

I had the opportunity to experience Ph.D. life in two esteemed universities, the University of Southern California and the University of Minnesota. I hold both dear to my heart. I cherish the eternal sunshine of Los Angeles and will never forget Minnesota's vibrant autumns adorned with falling leaves and its serene winters blanketed in snow.

System research has taught me that every design choice comes with a price, just like every decision in life. I have never doubted my choice to pursue a Ph.D., but it came with its own costs.

During my Ph.D. study, my grandfather, Rongren Xu, passed away in 2022, and my grandmother, Chunxiang Du, passed away just one month ago as I write this paragraph. Unfortunately, due to various reasons, I was unable to return home to bid them farewell – a profound regret that I carry with me. This dissertation is in loving memory of them.

Finally, I would like to express my deepest gratitude to my beloved parents, Xinghua Xu and Longman Zhang, for their unconditional love, support, and encouragement throughout my life. They are the root of my courage to strain at the oars. This dissertation is dedicated to them.

Table of Contents

List of Tables

List of Figures

Abstract

Optimizing Emerging Multimedia Systems:

A Principled Approach to Enhancing Quality-of-Experience

by

Anlan Zhang

Chair: Feng Qian

Recent years have witnessed a surge in innovative networked multimedia applications, which rely on streaming multimedia content among various entities (*e.g.,* between servers/edge nodes and clients, among end users, *etc.*). Despite their potentials, these applications still face challenges in providing a consistently high quality-of-experience (QoE): the evolution of existing network infrastructures worldwide lags behind the fast growth of multimedia traffic over the Internet, while each application exhibits unique characteristics in terms of data representation, resource requirements, and QoE assessment. My dissertation is dedicated to addressing the above challenges, with the goal of boosting the QoE of emerging networked multimedia applications on existing network infrastructures through principled system design: developing adaptive, robust, and resource-efficient multimedia systems through innovative codecs, cross-layer optimizations, and user-centered QoE metrics, backed by solid prototyping and real-world evaluation. In my dissertation, I demonstrate how the principle can be applied and its effectiveness through in-depth

studies of four representative applications: volumetric video streaming, general mobile immersive content delivery, image live co-editing, and low-bitrate video conferencing. Specifically, I propose four novel systems and frameworks: YuZu, the first volumetric video streaming system enhanced by 3D super resolution; Habitus, the first software framework aimed at optimizing the upper-layer network protocol stack for immersive content delivery and, more broadly, metaverse applications; Alice, a cross-platform compression adaptation framework for low-latency image live co-editing under fluctuating network and computational resources; and NIER, a practical neural-enhanced low-bitrate video conferencing solution suitable for a wide range of usage scenarios, particularly over challenging or metered networks. Through careful design, solid prototyping, and extensive evaluation, I show that all proposed systems and frameworks significantly improve the QoE of their corresponding networked multimedia applications compared to the baseline and state-of-the-art systems. I believe that my studies on the above applications, especially the principled approach, provide insights for a broader range of novel networked multimedia applications and can be easily adapted and applied to their system design and optimization.

Chapter 1

Introduction

Recent years have witnessed a surge in innovative multimedia content and networked multimedia applications. For instance, volumetric video [257, 255, 254, 256], a novel type of immersive content, has found applications across diverse domains, including healthcare, education, and entertainment. Similarly, remote collaborative multimedia applications, such as image live co-editing [159] and video conferencing [268, 267], have garnered increasing attention since the onset of the COVID-19 pandemic. At their core, these applications rely on streaming multimedia content among various entities (*e.g.,* between servers/edge nodes and clients, among end users, and *etc.*). Despite their potentials, it remains challenging for these emerging networked multimedia applications to provide a consistently high quality-of-experience (QoE) when streaming their multimedia data over existing network infrastructures, due to two major reasons:

From the perspective of network infrastructures, their evolution – namely technical advancements and commercial deployment – lags behind the fast growth of multimedia traffic over the Internet. First, while new networking technologies such as 5G are being rapidly rolled out worldwide, there is limited research on how these technologies can be effectively leveraged to

support innovative networked multimedia applications, such as immersive content delivery. Second, network conditions (*e.g.,* bandwidth and packet loss rate) are fluctuating and difficult to predict or even estimate accurately in the wild. This presents a critical and common challenge across various networked multimedia applications, *i.e.,* how to adapt to dynamic network conditions. Furthermore, the current deployment of network infrastructure remains geographically imbalanced. While developed regions such as North America and Europe benefit from stable and high-speed Internet access, many areas still suffer from poor broadband connectivity, characterized by low bandwidth and high packet loss. In such regions, everyday applications like video conferencing often deliver suboptimal QoE or may fail to function altogether.

From the perspective of networked multimedia applications, each exhibits distinctive characteristics in terms of data representation, resource requirements, and QoE assessment. Specifically, immersive content delivery systems demand substantial network resources (*e.g.,* hundreds of Mbps or even Gbps for high-quality volumetric videos); image live co-editing systems require low-latency and lossless image edit transmission among multiple users; and video conferencing applications must remain resilient under challenging network conditions (*e.g.,* low bandwidth and high packet loss rate). These inherent differences introduce unique challenges that necessitate in-depth investigation and tailored design solutions for each application.

My thesis is dedicated to address the above challenges, with the goal of *boosting the quality-of-experience (QoE) of emerging networked multimedia applications on existing network infrastructures through principled system design*: developing adaptive, robust, and resource-efficient multimedia systems through innovative codecs, cross-layer optimizations, and user-centered QoE metrics, backed by solid prototyping and real-world evaluation. To explain, the principle to enhancing QoE includes: (1) exploring innovative coding schemes for multimedia content (§3, §5, §6); (2)

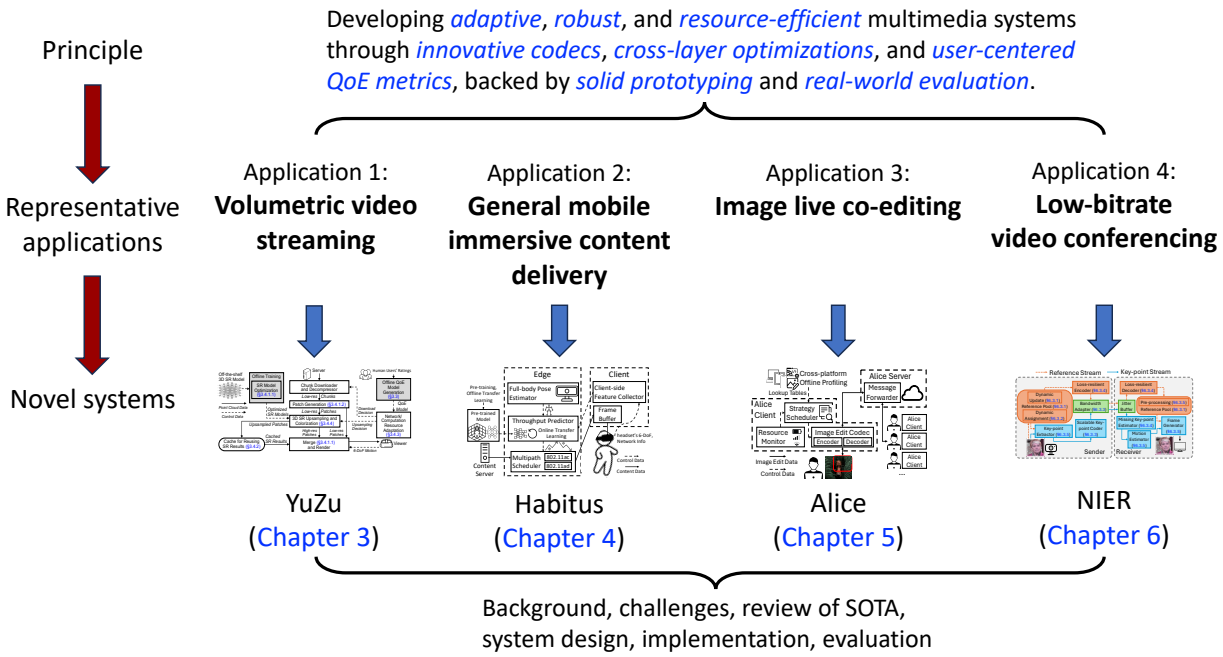


Figure 1.1: Overview of this dissertation.

enhancing adaptability and robustness for streaming multimedia data under fluctuating network conditions (§3, §4, §5, §6); (3) conducting user studies to understand real users' QoE expectations for various applications (§3, §4, §5, §6); (4) leveraging cross-layer design (§4, §6) and (5) seeking co-existence with commodity network infrastructures and minimizing deployment effort (§3, §4, §5, §6). Specifically, I demonstrate how the principle can be applied and its effectiveness through in-depth studies of four emerging and representative networked multimedia applications: *volumetric video streaming*, *general mobile immersive content delivery*, *image live co-editing*, and *low-bitrate video conferencing*. I believe that my studies with the above applications, especially the principled approach, provide insights for a broader range of novel networked multimedia applications and can be easily adapted and applied to their system design and optimization. The overview of my dissertation is illustrated in Figure 1.1, and I elaborate on its contributions in the following four sections.

1.1 Neural-enhanced Volumetric Video Streaming

Differing from traditional 2D videos, volumetric videos provide true 3D immersive viewing experiences and allow viewers to exercise six degree-of-freedom (6DoF) motion. However, streaming high-quality volumetric videos over the Internet is extremely bandwidth-consuming. To improve the quality-of-experience (QoE) under limited bandwidth, prior work has mostly focused on viewport-adaptive streaming (*i.e.,* mainly streaming content that will appear in the viewport) [68, 102, 156]. However, they are ineffective when the entire scene falls inside the viewport. They also require 6DoF motion prediction that is unlikely to be accurate for fast motion. Some other proposals explored remote rendering [64, 165] (*e.g.,* having an edge node transcode 3D scenes into regular 2D frames). However, they require not only 6DoF motion prediction, but also edge/cloud-side transcoding that is difficult to scale.

In this work, we propose to leverage 3D super resolution (SR) to drastically increase the visual quality of volumetric video streaming. To accomplish this goal, we conduct deep intra- and inter-frame optimizations for off-the-shelf 3D SR models, and achieve up to 542× speedup on SR inference without accuracy degradation. We also derive a first Quality of Experience (QoE) model for SR-enhanced volumetric video streaming, and validate it through extensive user studies involving 1,446 subjects, achieving a median QoE estimation error of 12.49%. We then integrate the above components, together with important features such as QoE-driven network/compute resource adaptation, into a holistic system called YuZu that performs line-rate (at 30+ FPS) adaptive SR for volumetric video streaming. Our evaluations show that YuZu can boost the QoE of volumetric video streaming by 37% to 178% compared to no SR, and outperform existing viewport-adaptive solutions by 101% to 175% on QoE.

To summarize, we make the following contributions: (1) We build an empirical QoE model for SR-enhanced volumetric videos, and validate it through large-scale user studies involving 1,446 participants. We build our models using volumetric content of single/multiple human portraits, a major application of volumetric video streaming. Note that the model can be applied to non-SR volumetric videos belonging to the same genre, with an SR ratio of 1; (2) We propose and design YuZu, an SR-enhanced, QoE-aware volumetric video streaming system; and (3) We implement YuZu, and conduct extensive evaluations for its QoE improvement and runtime performance.

1.2 Boosting Mobile Immersive Content Delivery through Full-body Pose Tracking and Multipath Networking

Delivering high-quality immersive content such as volumetric videos and virtual/mixed reality requires tremendous network bandwidth. Millimeter Wave (mmWave) radios such as 802.11ad/ay and mmWave 5G can provide multi-Gbps peak bandwidth, making them good candidates. However, mmWave is vulnerable to blockage/mobility and its signal attenuates very fast, posing a major challenge to mobile immersive content delivery systems where viewers are in constant motion and the human body may easily block the line-of-sight. To overcome this issue, existing systems take three categories of approaches: (1) *Improving the PHY layer* [54, 233, 226]; (2) *Enhancing line-of-sight (LoS)* [221]; and (3) *Using specialized equipment* [2]. However, they all have limitations. Engineering the PHY layer alone is inadequate to handle, *e.g.,* the throughput fluctuation incurred by viewers' fast motion. Mounting the radio overhead may still incur frequent non-line-of-sight (NLoS) blockages (*e.g.,* when the viewer looks up/down, raises arms, or passes

through an obstacle). Adding reflectors to combat NLoS raises the deployment bar and is incompatible with commodity mmWave protocols. Even in the absence of blockage, highly dynamic user mobility can still cause significant performance drops of mmWave [201, 183, 5].

In this work, we investigate two under-explored dimensions. First, we use the combination of a viewer's full-body pose and the network information to predict mmWave performance as the viewer exercises six-degree-of-freedom (6-DoF) motion. We apply both offline and online transfer learning to enable the prediction models to react to unseen changes. Second, we jointly use the omnidirectional radio and mmWave radio available on commodity mobile devices to deliver immersive data. We integrate the above two features into a user-space software framework called Habitus, and demonstrate how it can be easily integrated into existing immersive content delivery systems to boost their network performance, which leads to up to 72% of quality-of-experience (QoE) improvement.

Habitus represents to our knowledge a first software framework aiming at optimizing the upper-layer network protocol stack for immersive content delivery (and metaverse applications in general). This work makes three-fold contributions: the design of the Habitus framework; its implementation, evaluation, and integration into two volumetric content delivery systems; and the release of data [67] (802.11ac/ad performance correlated with full-body motion) and source code [66].

1.3 Low-latency Image Live Co-editing via Adaptation

Image live co-editing (LCE), which allows users to edit a shared image concurrently and remotely, is rising in popularity. However, fluctuating resources (*i.e.,* bandwidth and computation), as well

as varying degrees of edit complexity, make it challenging to achieve low-latency image live co-editing, which drastically degrades the user experience. Many networked multimedia applications, such as real-time communication and video streaming, face similar challenges as those mentioned above. Existing approaches to reducing end-to-end multimedia transmission latency can be categorized into three main strategies: (1) *Reducing in-network queuing delay* [133, 223, 204]; (2) *Adaptive streaming* [197, 166, 68]; and (3) *Accelerating compression* [191]. However, they all have limitations in the context of image live co-editing. *Reducing in-network queuing delay* can be helpful, but overall latency may still be constrained by network limitations due to the low compression ratio of lossless compression. *Adaptive streaming* relies on lossy compression, which is incompatible with image LCE's requirement for lossless compression, limiting its applicability. *Accelerating compression* through extensive parallelization demands high on-device computational power, which may not always be available. Additionally, network constraints may still impose latency limits due to the restricted compression ratio of lossless methods.

In this work, to achieve low-latency image LCE, we propose Alice, a cross-platform compression adaptation framework that incorporates three core designs. First, Alice leverages both *data-based* (*i.e.,* sending compressed pixels) and *operation-based* (*i.e.,* sending image editing operation APIs and corresponding parameters) approaches for image edit transmission. Second, Alice combines diverse modern lossless compression techniques and their various configurations to enhance the adaptability of *data-based* transmission. Third, Alice features a lookup table (LUT)-based decision framework to determine the best transmission strategy for image edits in real time. We implement Alice and integrate it into our image LCE testbed. Our extensive evaluation shows that, compared to the baselines using a fixed transmission strategy, Alice achieves up to 95% latency reduction with negligible overhead.

To summarize, our contributions include: (1) To the best of our knowledge, this is the first study to focus on low-latency image live co-editing, addressing the problem from a system perspective; (2) The design of Alice, including its hybrid transmission strategy and the LUT-based adaptation algorithm; and (3) The implementation, integration, and thorough evaluation of Alice on our self-developed testbed.

1.4 Practical Neural-enhanced Low-bitrate Video Conferencing

Low-bitrate video conferencing benefits multiple stakeholders: streaming platforms spend less on network infrastructures; cellular providers see reduced peak-hour traffic; mobile customers pay less over metered links; and most importantly, end users perceive better quality-of-experience (QoE) under challenging network conditions. The existing solution for achieving low-bitrate video conferencing with decent QoE is to stream low-resolution video frames using traditional codecs [172, 203, 20], and then apply image enhancement techniques like super-resolution [195] at the receiver to boost visual quality. One issue of this approach is that, for efficiency, all traditional codecs incur high temporal dependency; *i.e.,* they produce P-frames whose decoding depends on prior I- or P-frames. As a result, a single packet loss can lead to the undecodability of multiple consecutive frames. Note that packet losses are prevalent in challenging network conditions – a key usage scenario of low-bitrate video streaming. There are a few techniques to counteract packet losses for real-time video communication, such as retransmissions, forward error correction (FEC) [141, 180], error concealment [228], and advanced loss-resilient neural codecs [36,

107]. However, they suffer from various limitations such as prolonged latency, extra bandwidth cost, low compression efficiency, and prohibitively high compute overhead, respectively.

In this work, we develop NIER, a practical low-bitrate video conferencing solution. It can adaptively maintain a low bitrate (*e.g.,* 10–100 Kbps) with reasonable visual quality while being robust to packet losses. Satisfying these design requirements makes NIER suitable for a wide range of usage scenarios, in particular over challenging/metered networks. Under the hood, NIER leverages key-point-based deep image animation (DIA) as a key building block, where the sender transmits sparse key-points alongside a reference image, and the receiver reconstructs the original video frames by animating the reference image using the key-points' motion. To make DIA practical, NIER addresses a series of challenges in networking and system dimensions, including robustly updating reference frames, adapting to fluctuating bandwidth, handling varying packet loss rates, and achieving line-rate frame processing on commodity client devices. Our extensive evaluations (including an IRB-approved user study involving 20 participants) demonstrate that NIER considerably outperforms several baseline solutions (traditional video codecs, super-resolution-enhanced video conferencing, forward error coding (FEC), loss-resilient neural codec, and naive application of key-point-based DIA) in terms of end-to-end latency, decodable frame ratio, frame rate, video quality, and/or users' quality-of-experience (QoE).

NIER is to our knowledge the first practical low-bitrate video conferencing solution enhanced by key-point-based DIA. Our contributions include: (1) the design of NIER; (2) the implementation and end-to-end evaluation of NIER; and (3) the extensive comparison between NIER and the state-of-the-art low-bitrate and loss-resilient video conferencing solutions.

1.5 Thesis Organization

This dissertation is structured as follows. Chapter 2 provides sufficient background of the four networked multimedia applications mentioned above. In Chapter 3, we study on-demand volumetric video streaming, and propose YuZu, a first-of-its-kind system that applies 3D super resolution to enhance the quality-of-experience (QoE) of volumetric video streaming over the Internet. We then extend our focus to broader immersive content (*e.g.,* volumetric videos, VR/MR), and Chapter 4 presents Habitus, a generic framework developed by us for mobile immersive content delivery, enhanced by multipath networking over omnidirectional (*e.g.,* 802.11ac) and mmWave radios (*e.g.,* 802.11ad), and proactive mmWave throughput prediction. Chapter 5 introduces our research on image live co-editing (LCE), another emerging networked multimedia application. In particular, we develop Alice, a cross-platform compression adaptation framework to realize low-latency user experience for LCE under fluctuating network/computation resources. Next, in Chapter 6, we switch to an everyday application, video conferencing, and propose NIER, a practical neural-enhanced video conferencing solution that is suitable for a wide range of usage scenarios, in particular over challenging/metered networks. Finally, we summarize all related work in Chapter 7 before concluding the thesis in Chapter 8.

Chapter 2

Background

This chapter provides a sufficient background of the four emerging networked multimedia applications studied in this dissertation: *volumetric video streaming*, *mobile immersive content delivery*, *image live co-editing*, and *low-bitrate video conferencing*, including the challenges they face, a brief overview of existing solutions, the core technologies we adopt to boost their quality-of-experience (QoE) as well as the challenges of applying these technologies.

2.1 Volumetric Video Streaming

Volumetric video is an emerging type of multimedia content. Unlike traditional videos and 360° panoramic videos [71, 166] that are 2D, every frame in a volumetric video consists of a 3D scene represented by a point cloud or a polygon mesh. The 3D nature of volumetric video enables viewers to exercise six degree-of-freedom (6DoF) movement: a viewer can not only "look around" by changing the yaw, pitch, and roll of the viewing direction, but also "walk" in the video by changing the translational position in 3D space. This leads to a truly immersive viewing experience.

Schemes	Refs	Advantages (⊕) and Disadvantages (⊖)
Direct Streaming	N/A	⊕ Easy to implement, best QoE (if bandwidth is sufficient). ⊖ Highest network bandwidth (BW) usage.
Direct + VA	[68, 102]	⊕ Lower BW usage. ⊖ BW saving depends on user's motion, QoE depends on motion prediction.
Direct + SR	YuZu	⊕ Good QoE, further lower BW usage, adaptively trades compute resource for BW. ⊖ Requires training.
Remote Rendering	[64, 165]	⊕ Lowest BW usage. ⊖ QoE depends on motion prediction, need edge support (poor scalability).

Table 2.1: Four categories of volumetric video streaming approaches (VA = Viewport Adaptation; SR = Super Resolution).

As the key technology of realizing telepresence [154], volumetric video has registered numerous applications. They can be viewed in multiple ways: through VR/MR (virtual/mixed reality) headsets or directly on PCs (similar to how we play 3D games).

Despite the potentials, streaming volumetric videos over the Internet faces a key challenge of high bandwidth consumption. High-quality volumetric content requires hundreds of Mbps bandwidth [68, 255]. To improve the quality-of-experience (QoE) under limited bandwidth, prior work has mostly focused on viewport-adaptive streaming (*i.e.,* mainly streaming content that will appear in the viewport) [68, 102, 156]. However, they are ineffective when the entire scene falls inside the viewport. They also require 6DoF motion prediction that is unlikely to be accurate for fast motion. Some other proposals explored remote rendering [64, 165] (*e.g.,* having an edge node transcode 3D scenes into regular 2D frames). However, they require not only 6DoF motion prediction, but also edge/cloud-side transcoding that is difficult to scale, as summarized in Table 2.1.

In Chapter 3, we employ a different and orthogonal approach toward improving the QoE of volumetric video streaming through *3D super resolution* (3D SR). SR was initially designed for improving the visual quality of 2D images [33, 244]. Recently, researchers in the computer vision community developed SR models for point clouds [106, 229, 234, 249]. This inspires us to employ SR for volumetric video streaming, as each frame of a volumetric video is typically either a point cloud or a 3D mesh.* We present more details and a case study of applying 3D SR to volumetric video streaming in the following, which motivates our design of YuZu in Chapter 3.

2.1.1 Applying 3D SR to Volumetric Video Streaming: A Case Study

Recently, the computer vision community extended SR to *static* point clouds [106, 229, 234, 249]. When applied to a video v, SR trains offline a deep neural network (DNN) model M that *upsamples* low-resolution frames $L(v)$ to high-resolution ones $H(v)$, using the original (high-resolution) frames $F(v)$ for training. In the online inference, the server sends M and $L(v)$ to the client, which infers $H(v) = M(L(v))$. SR leverages the overfitting property of DNN to ensure that $H(v)$ is highly similar to $F(v)$. It achieves bandwidth reduction (or QoE improvement when bandwidth remains the same) since the combined size of M and $L(v)$ is much smaller than $F(v)$.

We start with a straightforward approach: applying PU-GAN [106], a state-of-the-art 3D SR model, to upsample every point cloud frame of a volumetric video. PU-GAN operates by dividing the entire point cloud of a frame into smaller *patches*, each consisting of a subset of points. Both SR training and inference are performed on a per-patch (as opposed to a per-frame) basis, *i.e.,* each patch is upsampled individually. Its DNN model is based on a generative adversarial

*We focus on point-cloud-based volumetric videos in this work, but the key concepts of YuZu also apply to mesh-based volumetric videos.

network (GAN) and realizes three key stages: feature extraction, feature expansion, and point set generation.

We next describe a case study using PU-GAN to motivate YuZu. Our testing video was captured by three depth cameras. It has 3,622 frames, each consisting of ~100K points depicting a performing actor. We refer to this video as *Lab*. We use all its frames to train a PU-GAN model. We set the SR ratio (*i.e.,* upsampling ratio) to 4, making the input and output point clouds consist of roughly 25K and 100K points, respectively.

We have both positive and negative findings from this case study. On the positive side, the model can accurately reconstruct each individual frame, *i.e.,* each upsampled point cloud is highly similar to the original one in terms of the geometric structure, as quantified by the Earth Mover's Distance (EMD [179]):

$$\mathcal{L}_{EMD}(I, G) = \min_{\phi: I \to G} \frac{1}{|I|} \sum_{x \in I} \|x - \phi(x)\|_2 \tag{2.1}$$

where I and G are the upsampled point cloud and the ground truth, respectively; $\phi : I \to G$ is a bijection from the points in I to those in G. The average EMD value across all frames is 1.47cm, which confirms good upsampling accuracy [106]; it is also verified by our IRB-approved user studies (§3.3.2). Also encouragingly, we find that SR indeed achieves significant bandwidth savings. For this 2-minute video, the compressed sizes of $F(v)$, M, and $L(v)$ are 1.40 GB, 560 KB, and 0.36 GB, respectively, leading to a bandwidth reduction of 74.2%.

Despite the above encouraging results, we notice three major issues from the above case study.

• **A Lack of Quality-of-Experience (QoE) Model.** For traditional 2D video streaming, there exist numerous studies on modeling the viewer's QoE [18, 248, 216]. In contrast, volumetric videos

are still in their infancy. There is a lack of generic QoE models that researchers can leverage, not to mention a lack of understanding of how SR impacts QoE.

• **Unacceptably Poor Runtime Performance.** 3D SR models are computationally much more heavyweight than 2D SR models. When applying PU-GAN to the above video, the runtime performance is extremely poor. On a machine with an NVIDIA 2080Ti GPU, the upsampling FPS is only 0.1, far below the desired FPS of at least 30. Besides, the GPU memory usage of PU-GAN is 7GB (out of the 11GB available memory of 2080Ti). This is one reason why all the off-the-shelf 3D SR models operate on a per-patch basis, as this saves memory compared to processing a full frame.

• **No Color Support.** We find that no existing 3D SR model can restore the color information of upsampled point cloud.

Note that the last two limitations are common in that they also apply to all other 3D SR models for point clouds that we have examined, such as MPU [229] and PU-Net [249].

2.2 Mobile Immersive Content Delivery

Immersive content, such as virtual/mixed reality (VR/MR) and volumetric videos, allows viewers wearing VR/MR headsets to exercise six-degree-of-freedom (6-DoF) motion (yaw, pitch, roll, X, Y, Z), offering a truly engaging experience [68, 135, 140, 103]. Networked immersive content delivery systems require tremendous network resources (*e.g.,* hundreds Mbps or even Gpbs for high-quality volumetric videos [68, 254, 102]). This poses a major challenge for mobile immersive content delivery systems, which use wireless radios instead of HDMI/USB cables [219, 218] for content delivery.

Recent advances in millimeter wave (mmWave) radio technologies make it feasible to transmit immersive content at a multi-Gbps data rate. mmWave protocols such as 802.11ad [147] (802.11ay [55] in the future) and mmWave 5G [70] have been commercialized on commodity mobile devices. Despite its high data rate, compared to omnidirectional radio, mmWave radio is much more vulnerable to blockage/mobility and its signal attenuates much faster [147]. This creates a major issue for immersive applications where viewers are in constant motion and the human body may easily block the line-of-sight. To overcome this issue, existing systems take three categories of approaches.

• *Improving the PHY layer.* Numerous studies have been conducted on the mmWave radio in general, such as improving MIMO [54] and beamforming [233, 226].

• *Enhancing line-of-sight (LoS).* Some off-the-shelf commercial products [221] mount the mmWave radio on top of a VR headset to avoid blockages.

• *Using specialized equipment.* Some prior research [2] proposes to deploy multiple reflectors paired with a custom PHY protocol design to improve VR performance over mmWave.

The above approaches help but all have limitations. Engineering the PHY layer alone is inadequate to handle, *e.g.,* the throughput fluctuation incurred by viewers' fast motion. Mounting the radio overhead may still incur frequent non-line-of-sight (NLoS) blockages (*e.g.,* when the viewer looks up/down, raises arms, or passes through an obstacle). Adding reflectors to combat NLoS raises the deployment bar and is incompatible with commodity mmWave protocols. Even in the absence of blockage, highly dynamic user mobility can still cause significant performance drops of mmWave [201, 183, 5].

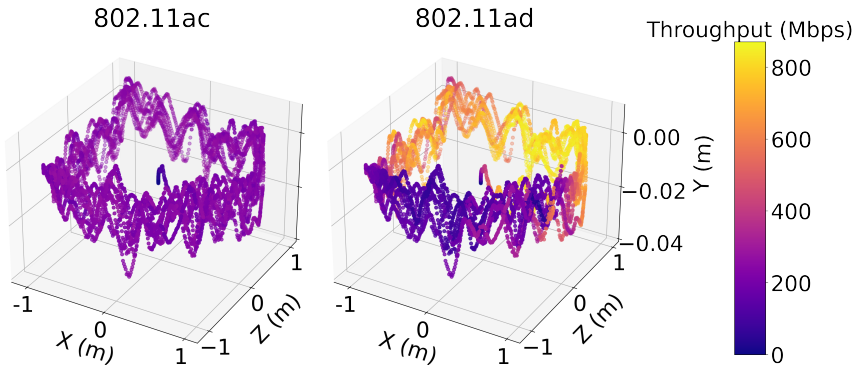


Figure 2.1: Spatial heatmap of 802.11ac/ad throughput in a room.

In the following, we present a case study of applying mmWave to immersive content delivery using mobile volumetric video streaming as an example, which motivates our design of Habitus in Chapter 4.

2.2.1 Applying mmWave to Immsersive Content Delivery: A Case Study

Despite their great potentials on boosting immersive content delivery [68, 102], mmWave signals suffer from increased attenuation, mobility, and blockages [147]. In contrast, 802.11ac operates at the 5GHz band with omnidirectional signal propagation, providing lower but more stable throughput than 802.11ad. In a case study conducted in a personal office (Figure 4.3), we investigate how the PHY properties of ac and ad affect the QoE of immersive content delivery. Using a smartphone [176] mounted on the user's head and a volumetric video streaming application [68], we measure the QoE while the user walks around the room. Results show that using 802.11ad greatly improves the content quality by 113%, but also hugely increases the video stall by 502% due to its fluctuating throughput under user mobility (Figure 2.1). Our case study reveals *that ac and ad have distinct network performance due to their complementary PHY properties, and motivates us to strategically combine them to enhance the QoE of immersive content delivery.*

A plethora of studies focus on improving the communication quality of mmWave on the PHY layer [54, 233, 226], while ignoring contextual information for immersive content delivery where a viewer's full body is constantly in motion. On the other side, although solutions with application domain knowledge (*e.g.,* [2, 221]) have shown some effectiveness, they are either incompatible with existing PHY-layer protocols [2] or still suffer from significant LoS blockages [221]. Such a gap motivates us to propose solutions that *judiciously leverage viewer's full-body motion to facilitate mmWave performance forecast, while being compatible to commercial mmWave protocols (802.11ad, mmWave 5G/6G,* etc.*) and easily integrable into diverse immersive applications.*

2.3 Image Live Co-editing

Live co-editing (LCE) applications [60, 155, 50, 47] have become crucial for boosting workplace productivity. Emerging image LCE tools [159], which are newer and less studied compared to their text-based counterparts, extend these capabilities to image editing. Ideally, image LCE tools should possess three essential features to enhance work flexibility and enable efficient collaboration from anywhere at any time: *lossless* information transmission, *low-latency* interactions, and *scalability*. However, image LCE systems are still in their early stages and lack substantial research. Chapter 5 conducts a latency-focused study of cloud-based image LCE systems. We focus on achieving a low-latency user experience, where latency is defined as *the elapsed time between when a user makes an edit to a shared image on their local machine and when the edit appears on the devices of other users.*

Many networked multimedia applications, such as video streaming and real-time communication, face similar challenges as those mentioned above. Existing approaches to reducing end-to-end multimedia transmission latency can be categorized into three main strategies: (1) *Reducing in-network queuing delay.* Numerous studies [133, 223, 204] focus on minimizing queuing delays on the network side; (2) *Adaptive streaming.* Adaptive bitrate (ABR) algorithms are widely used to ensure timely video delivery under fluctuating bandwidth conditions [197, 166, 68]; and (3) *Accelerating compression.* Some studies [191] reduce image compression latency by leveraging high-performance computing units, such as GPUs.

The above approaches provide benefits but have limitations in the context of image live co-editing. *Reducing in-network queuing delay* can be helpful, but overall latency may still be constrained by network limitations due to the low compression ratio of lossless compression. *Adaptive streaming* relies on lossy compression, which is incompatible with image LCE's requirement for lossless compression, limiting its applicability. *Accelerating compression* through extensive parallelization demands high on-device computational power, which may not always be available. Additionally, network constraints may still impose latency limits due to the restricted compression ratio of lossless methods.

We next provide an overview of the basic cloud-based image LCE system, and the resource uncertainty in image LCE, which motivates us to develop Alice in Chapter 5.

2.3.1 Basic Image LCE System Overview

We begin by introducing the cloud-based image LCE system, which serves as the foundation of this work. As shown in Figure 2.2, the system consists of a server and multiple clients. All shared

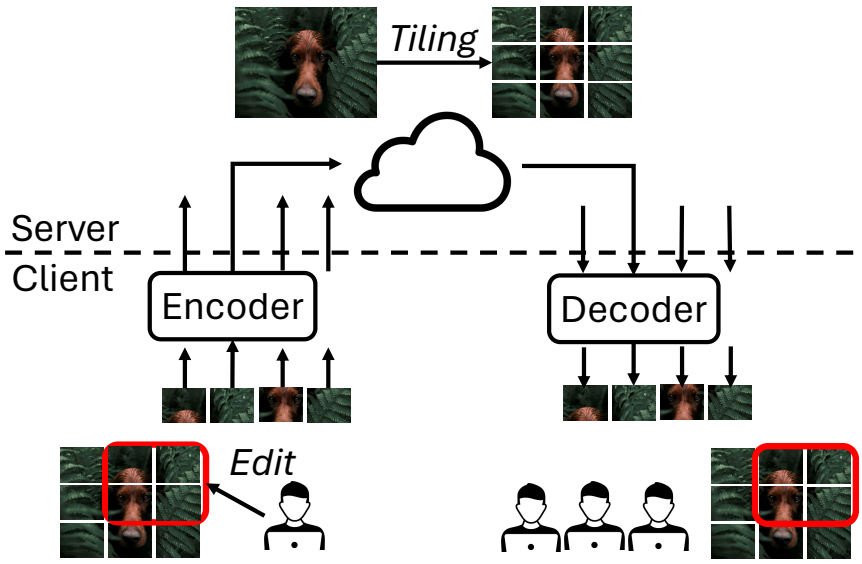


Figure 2.2: The image LCE system using a server-client architecture.

images are initially stored on the LCE server. When a new image LCE session starts, the server distributes the shared image to all participating clients. Once users have local copies, they can begin editing. LCE clients capture user edits on the local image, send these edits to the server, and apply edits received from the server to their local copies. The LCE server manages conflicts arising from concurrent edits by different clients and distributes valid edits to all users. [†]

The image LCE system transmits image edits by sending their pixel data over the network, optionally compressed using lossless techniques. We refer to this approach as *data-based* transmission. Specifically, a shared image is spatially segmented into smaller, non-overlapping tiles, each with a lower resolution (*e.g.*, 512×512, 1024×1024, *etc.*), and a single channel, unlike the multi-channel full image. Compression and transmission of image edits occur on a per-tile basis, involving only the tiles affected by the edit (*e.g.*, the red area in Figure 2.2). This design is inspired by viewport-adaptive streaming in immersive video systems [166, 68], which transmits only the content within the viewer's viewport to conserve bandwidth. In the context of image

[†]We assume that conflict management is efficient and does not introduce a significant end-to-end latency bottleneck. Therefore, we do not discuss its details in this work.

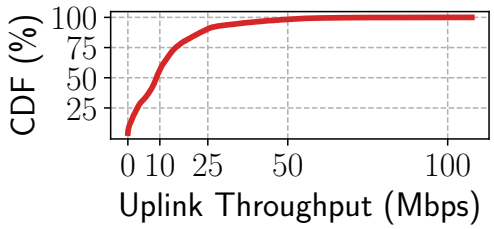


Figure 2.3: CDF of FCC mobile uplink throughput in Jan. 2023 [131].

LCE, the area affected by the edit serves as a conceptual “viewport" for transmission. Our user study results (§5.3) further validate this design choice.

2.3.2 Resource Uncertainty In Image LCE

Achieving consistently low-latency performance in image LCE systems is challenging, especially in mobile use cases, due to the following factors: (1) Unreliable wireless communication: Mobile networks are inherently unstable, with frequent oscillations and reliability issues across all available infrastructures [70, 108, 78, 240]. For example, Figure 2.3 presents the FCC’s mobile broadband uplink throughput distribution in 10 major U.S. cities (January 2023) [131]. The average and median throughputs are 11.41 Mbps and 8.95 Mbps, respectively, with a standard deviation of 11.54 Mbps. For 90% of the time, uplink throughput remains below 25 Mbps; (2) Heterogeneous computational resources: Image LCE clients run on diverse platforms (*e.g.,* laptops, tablets, and web), leading to significant variations in computational capabilities; and (3) Varying complexities of image editing operations: Image edits differ widely in computational intensity, making it difficult to design a one-size-fits-all solution. These *motivates us to enhance the adaptability of image LCE systems to better cope with varying resource constraints.*

2.4 Low-bitrate Video Conferencing

Video conferencing requires a substantial amount of network bandwidth. For example, Zoom requires at least 1.2 Mbps bandwidth for both uplink and downlink in a 1-on-1 video call at 720p [267], which amounts to ~1.05 GB of data for a one-hour session. *Low-bitrate video conferencing* thus benefits multiple stakeholders: streaming platforms spend less on network infrastructures; cellular providers see reduced peak-hour traffic; mobile customers pay less over metered links; and most importantly, end users perceive better quality-of-experience (QoE) under challenging network conditions.

A promising approach to realize low-bitrate video conferencing with decent QoE is to stream low-resolution video frames using traditional video codecs such as H264 [172], HEVC [203], and VPX [20], and then apply image enhancement techniques like super-resolution (SR) [195] at the receiver to boost visual quality. One limitation of this approach is that, for efficiency, all traditional codecs incur high temporal dependency; *i.e.,* they produce P-frames whose decoding depends on prior I- or P-frames. As a result, a single packet loss can lead to the undecodability of multiple consecutive frames. Note that packet losses are prevalent in challenging network conditions – a key usage scenario of low-bitrate video streaming.

There are a few techniques to counteract packet losses for real-time video communication, such as retransmissions, forward error correction (FEC) [141, 180], error concealment [228], and advanced loss-resilient neural codecs [36, 107]. However, they suffer from various limitations such as prolonged latency, extra bandwidth cost, low compression efficiency, and prohibitively high compute overhead, respectively.

In Chapter 6, we develop a practical low-bitrate video conferencing solution, referred to as NIER, by leveraging a technique called *key-point-based deep image animation* (DIA) as a key building block.

In the following, we provide more details of key-point-based DIA, and also present the current naive key-point-based DIA enhanced low-bitrate video conferencing solution, which motivates our design of NIER in Chapter 6.

2.4.1 Key-point-based Deep Image Animation

The deep image animation (DIA) technique [242, 192, 227] was originally developed to animate a static image (*e.g.,* a static human portrait) by imitating the motion and deformation of another video clip (*e.g.,* a video of a talking person). The static image to animate is denoted as the "reference frame" f^R, while each frame i in the video clip is referred to as a "driving frame" f_i^D. For each frame f_i^D in the video clip, we can apply its motion and deformation to animate the static image f^R and generate a new frame f_i^G, by feeding f_i^D and f^R together into a DIA framework A, *i.e.,*

$$f_i^G = A(f^R, f_i^D) \tag{2.2}$$

A state-of-the-art type of DIA is the key-point-based DIA [192, 193, 194, 227], which typically consists of three key components: a key-point extractor K, a motion estimator E, and a frame generator G. All of them are deep neural networks (DNN) and trained in a self-supervised and end-to-end way.

The key-point extractor K extracts n (*e.g.,* 10) key-points from a frame f, where a key-point p_j is defined as its 2D coordinate (x_j, y_j) with some point-wise attributes a_j (*e.g.,* a 2×2 local

affine transformation matrix [104]). We follow this definition of key-point in the rest of this paper. The key-point extraction process is thus denoted as

$$K(f) = \{p_j \mid 1 \leq j \leq n\} \text{ and } p_j = \{(x_j, y_j), a_j\} \tag{2.3}$$

The motion estimator E takes the reference frame f^R along with its key-points $\{p_j\}^R$, and the key-points $\{p_j\}_i^D$ of a driving frame f_i^D, as the input. It outputs a motion field o_i (*i.e.,* a dense optical flow [23]), and an occlusion mask m_i, *i.e.,*

$$\{o_i, m_i\} = E(f^R, K(f^R), K(f_i^D)) \tag{2.4}$$

The frame generator G is a generative model containing three stages: feature encoding, feature warping, and feature decoding. It takes the reference frame f^R, and the motion estimation output $\{o_i, m_i\}$ of a driving frame f_i^D as its input and generates a new frame f_i^G, *i.e.,*

$$f_i^G = G(f^R, o_i, m_i) \tag{2.5}$$

Specifically, the feature encoder extracts the reference features from f^R, which are used to derive the features of f_i^G along with the motion field o_i through warping. The feature decoder then generates f_i^G with the warped features. However, due to the motion and deformation of the object, not all the features of f_i^G should be directly warped from the reference feature. Therefore, before feeding the warped features to the feature decoder, the occlusion mask m_i is applied on them to mask out those useless features.

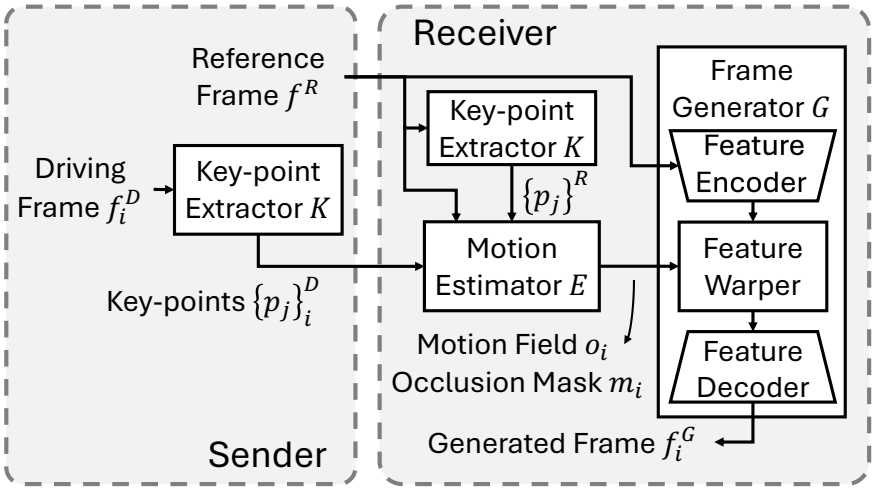


Figure 2.4: Naive solution for key-point-based DIA enhanced video conferencing.

2.4.2 Key-point-based DIA Enhanced Low-bitrate Video Conferencing: A Naive Solution

Several recent studies have explored applying the key-point-based DIA to low-bitrate video conferencing [4, 153]. They follow a common system design as shown in Figure 2.4: the key-point extractor K resides on the sender side, while the motion estimator E and frame generator G are on the receiver side. When a video call starts, the sender first sends a high-resolution image of their portrait (*i.e.,* the reference frame f^R in Figure 2.4) to the receiver. After that, the sender extracts the key-points $\{p_j\}_i^D$ from each newly captured frame (*i.e.,* the driving frame f_i^D in Figure 2.4), and transmits them to the receiver. The receiver uses the key-points $\{p_j\}_i^D$ and the reference frame f^R to generate a frame f_i^G and then render it to the user. This approach achieves a low bitrate since the size of the key-points $\{p_j\}_i^D$ is much smaller than the size of its frame f_i^D encoded by a traditional video codec [20, 203, 172].

Issues of the Naive Solution. We conduct a case study using the above architecture to motivate NIER. We train FOMM [193], a representative key-point-based DIA model, following

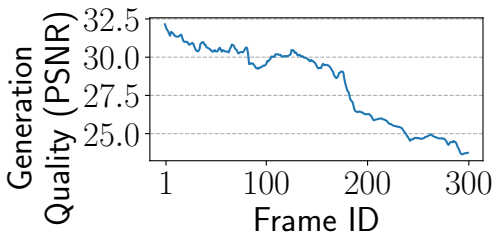


Figure 2.5: Quality drop over time for our test video (§2.4.2).

the setup in §6.5.1, and convert it to the ONNX [148] format for execution. We use one test video (§6.5.1), which has 1800 frames (1-minute playback at 30 FPS). We set up a 1-on-1 video call on a MacBook M1 2020 [10] with Neural Engine, using our trained model. We have both positive and negative findings. On the positive side, this approach indeed achieves low bitrate. Our 1-minute video only requires an average ~61.45 Kbps bitrate, consisting of a constant 56.25 Kbps for sending 10 key-points (FOMM [193]'s default setting, no entropy coding) at 30 FPS, and an average ~5.2 Kbps for sending the initial 35-KB reference frame. Meanwhile, we notice four major issues in this naive system architecture.

• **Issue 1: Quality Degradation over Time.** The naive solution only sends the reference frame once at the beginning. This can lead to a severe degradation of the frame generation quality over time. As Figure 2.5 shows, the generation quality in PSNR of our test video drops hugely from ~32.5 to ~23 after 10-second playback. Note that there is not much motion in our test video. In a scenario where more motion is involved and/or the session lasts a long time (*e.g.*, half an hour or even longer), the quality decrease can be more significant.

• **Issue 2: A Lack of Bandwidth Adaptation Mechanism.** The above design blindly sends key-points without awareness of the potential bandwidth fluctuation, especially for wireless networks [169, 70, 182]. This can cause network congestion and delay key-points delivery, which further leads to video freeze and degrades users' QoE (quality-of-experience).

• **Issue 3: Not Resilient to Packet Loss.** Though by encoding a frame as key-points, a lost packet only affects the decoding of one frame (we do not consider inter-frame coding since the key-points bitrate is already very low), users' QoE can still significantly drop when packet loss rate increases. Two common approaches, retransmission and FEC, can help but are sub-optimal in the low-bitrate scenario: they either incur a substantial delay or rely on an accurate packet loss rate estimation, and both need to send redundant data.

• **Issue 4: Poor Runtime Performance.** Key-point-based DIA is computationally heavy. On our test laptop, the frame generation of vanilla FOMM achieves only 11 FPS, with a high processing latency of 101 ms (sender + receiver side).

Chapter 3

YuZu: Neural-enhanced Volumetric Video Streaming

3.1 Introduction

Today's multimedia content is gaining not only higher resolutions, but also higher degrees of immersion, as demonstrated by, for example, the popularity of 360° panoramic videos [166]. Another emerging type of multimedia content is *volumetric videos* that bear even more immersion and user interactions. As described in §2.1, streaming high-quality volumetric videos over the Internet faces a key challenge of extremely high bandwidth consumption, and all existing solutions, such as viewport-adaptive streaming and remote rendering, have limitations.

In this chapter, we employ a different and orthogonal approach toward improving the QoE of volumetric video streaming through *3D super resolution* (3D SR). As introduced in §2.1, SR was initially designed for improving the visual quality of 2D images [33, 244], and the recent advancement in SR models for static point clouds inspires us to employ SR for volumetric video streaming, as each frame of a volumetric video is typically either a point cloud or a 3D mesh.* Although there have been recent successful attempts on applying SR to 2D video streaming [42, 91,

*We focus on point-cloud-based volumetric videos in this work, but the key concepts of YuZu also apply to mesh-based volumetric videos.

247], our case study in §2.1.1 shows that 3D-SR-enhanced volumetric video streaming is unique and challenging due to the following reasons.

• There is a fundamental difference between *pixel-based* 2D frames and volumetric frames consisting of *unstructured 3D points*, making processing volumetric videos (even without SR) vastly different from 2D videos.

• Due to its 3D nature, the computation overhead of 3D SR is very high. We apply off-the-shelf 3D SR models to volumetric videos [1], and find that the runtime performance of 3D SR is unacceptably poor – achieving only ~0.1 frames per second (FPS) on a PC with a powerful GPU. In contrast, 2D SR can achieve line-rate upsampling by simply downscaling the model [247], but we find that only doing model downscaling is far from being adequate for line-rate 3D SR (*i.e.,* at 30+ FPS).

• Given its recent debut, there lacks research on basic infrastructures such as tools and models supporting volumetric video streaming. For example, there is no QoE model for volumetric videos that can guide bitrate adaptation or critical SR parameter selection; the wide range of factors affecting the QoE make constructing such a model quite challenging.

• There are other practical challenges to overcome, such as a lack of color produced by today's 3D SR models.

To address the above challenges, we begin by developing to our knowledge a first QoE model for assessing SR-enhanced volumetric video streaming. The model takes into account a variety

of factors that may affect the QoE, such as video resolution (*i.e.,* point density)[†], viewing distance, upsampling ratio, SR-incurred distortion, and QoE metrics from traditional video streaming. We validate our model by conducting two IRB-approved user studies involving 1,446 voluntary participants from 40 countries, using a major genre of volumetric content, *i.e.,* portraits of single/multiple people. The validation results confirm its accuracy, with a median QoE estimation error of 12.49%. Our user studies offer definitive evidence that 3D SR can significantly boost the QoE of volumetric video streaming.

Next, we design, implement, and evaluate YuZu, which is to our knowledge the first volumetric video streaming system enhanced by SR. At its core, YuZu deeply optimizes the end-to-end upsampling pipeline in three aspects: *intra-frame SR*, *inter-frame SR*, and *network-compute resource management*, whose synergy helps drastically improve the runtime performance of SR while retaining the inference accuracy.

For **intra-frame SR**, our approaches are not limited to generic optimizations for deep learning models such as modifying SR models' structures for fast-paced SR. More importantly, we consider the factors that are unique to 3D SR and its data representation: we design a mechanism that leverages the low-resolution content (*i.e.,* the input to the SR model, which is typically discarded after being fed into the model) to reduce the SR model complexity; we also trim the pre-processing and post-processing stages of 3D SR and tailor them to volumetric video streaming. Note that these optimizations are generic, applicable to all the 3D SR models we have investigated [106, 229, 234, 249].

[†]The resolution of a point cloud is defined as its point density; the resolution of a volumetric video is the avg. resolution of its point cloud frames.

For **inter-frame SR**, YuZu speeds up SR by caching and reusing 3D SR results across consecutive frames. Realizing that none of the 2D inter-frame encoding techniques can be directly applied to volumetric videos, we design an effective inter-frame content reference scheme for SR-enhanced point cloud streams, followed by robust criteria determining whether SR results can be reused between two frames. We then extend reusing SR results from two to multiple consecutive frames through a dynamic-programming-based optimization. The synergy of the above intra- and inter-frame acceleration schemes fills the huge gap between off-the-shelf 3D SR models' performance and what is required for line-rate upsampling of point cloud streams.

YuZu further performs **network-compute resource management** through making judicious decisions about the quality level of the to-be-fetched content and its upsampling ratio. These two decision dimensions are subject to the dynamic network bandwidth and limited compute resources, respectively, which need to be jointly considered given their complex tradeoffs – a unique challenge compared to traditional adaptive bitrate (ABR) video streaming. YuZu takes a QoE-driven approach by maximizing the utility function derived from our QoE model. To solve the underlying optimization problem in real time, we develop a hybrid, two-stage algorithm that employs coarse-grained and fine-grained search at different time to efficiently find a good approximate solution. In addition, YuZu performs fast colorization of SR results through efficient nearest point search.

We implement the above components and integrate them into YuZu in 10,848 lines of code. Our extensive evaluations indicate that YuZu can achieve line-rate, adaptive, high-quality 3D SR. We highlight key evaluation results as follows.

• Our user study suggests that 3D SR can boost the volumetric video QoE by 37% to 178% compared to no SR.

• Our optimizations speed up 3D SR by 140× to 542× and reduce GPU memory usage by 68% to 90% with no accuracy degradation, compared to the vanilla SR models [106, 229].

• Compared to a recently proposed viewport-adaptive volumetric video streaming system [68], YuZu improves the QoE by 100.6% to 174.9%.

To summarize, we make the following contributions.

• We build an empirical QoE model for SR-enhanced volumetric videos, and validate it through large-scale user studies involving 1,446 participants. We build our models using volumetric content of single/multiple human portraits, a major application of volumetric video streaming. Note that the model can be applied to non-SR volumetric videos belonging to the same genre, with an SR ratio of 1.

• We propose and design YuZu, an SR-enhanced, QoE-aware volumetric video streaming system.

• We implement YuZu, and conduct extensive evaluations for its QoE improvement and runtime performance.

3.2 YuZu Overview

YuZu is to our knowledge the first SR-enhanced volumetric video streaming system. It streams video-on-demand volumetric content stored on an Internet server to client hosts. On the server side, the volumetric video is divided into *chunks* each consisting of a fixed number of frames (*i.e.,* point clouds encoded by schemes such as Octree [80, 132] and k-d tree [82, 114]). Each chunk is encoded into multiple versions with different resolutions (*i.e.,* point densities). The SR model training and volumetric content preprocessing (*e.g.,* patch reuse computation, see §3.4.2) are performed offline on the server side. Similar to a typical DASH server, the YuZu server is

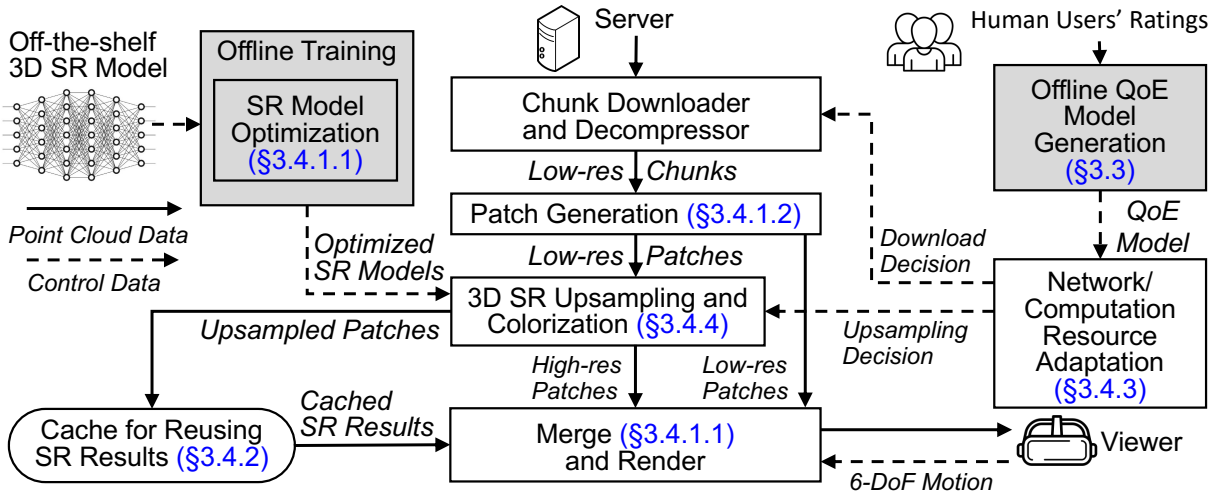


Figure 3.1: The system architecture of YuZu.

stateless (and thus scalable), and all the streaming logic runs on the client side. As shown in Figure 3.1, the client fetches from the server the video chunks, which can possibly be at a low resolution. Since 3D SR models typically operate on a per-patch basis, the client segments each frame into patches, upsamples them through 3D SR, efficiently colors them (§3.4.4), and renders them to the viewer.

To achieve line rate SR, YuZu employs novel optimizations tailored to SR-enhanced volumetric video streaming. Regarding *intra-frame optimizations*, off-the-shelf 3D SR models are strategically adapted; low-resolution patches before SR are properly leveraged instead of being discarded; and the patch generation is accelerated (§3.4.1). For *inter-frame optimizations*, previous SR results are judiciously reused (§3.4.2).

A crucial decision that YuZu must make is to determine what resolution (quality level) to fetch for each chunk, as well as which SR ratio to apply for upsampling each patch, subject to the resource constraints jointly imposed by the network and computation. YuZu addresses this through a principled, efficient, and QoE-driven discrete optimization framework (§3.4.3). The

framework utilizes a first-of-its-kind QoE model that we derive from ratings of 1,446 real users (§3.3).

3.3 QoE Model for Volumetric Videos

For SR-enhanced volumetric video streaming, its QoE is affected by a wide range of factors. The large space formed by these factors and their interplay make constructing QoE models much more challenging than conventional videos.

3.3.1 An Empirical QoE Model

We first enumerate factors that may affect the QoE for SR-enhanced volumetric video streaming. They are derived based on the domain knowledge of SR and our communication with other volumetric video viewers.

• **Point Density**. Similar to 2D image resolution, a 3D object with a higher point density (resolution) contains more details and thus offers a better QoE.

• **Viewing Distance.** As the viewing distance increases, a rendered 3D object becomes smaller in the displayed view, and is thus less sensitive to quality degradation.

• **SR Ratio and Distortion.** A higher SR ratio leads to a higher point density (and thus more QoE gain), but also potentially higher distortions (and thus more QoE loss).

• **Artifacts caused by Patches.** As described in §2.1.1, a typical 3D SR model operates by upsampling individual point subsets called patches. If patches within a frame have non-uniform qualities (caused by different SR ratios), the perceived QoE will be affected.

• **Invisibility due to Finite Viewport and Occlusion.** Due to the 3D nature of volumetric videos, a viewer can see only content that is inside the viewport and not occluded. Outside-viewport or occluded content brings no impact on the QoE.

• **QoE Metrics for Regular Video Streaming.** They include factors such as stall and inter-frame quality switches [248].

Next, we develop an empirical QoE model that considers the above factors. Since SR is performed on a per-patch basis, we first model the QoE for each individual patch as:

$$q_{i,j} = g(d_{i,j}, r_{i,j}, \delta_{i,j}) - h(EMD, \delta_{i,j}) \tag{3.1}$$

where $q_{i,j}$ is the quality of patch j in frame i; $d_{i,j}$ is the patch's original point density before SR; $\delta_{i,j}$ is the viewing distance to the patch; $r_{i,j}$ is the SR ratio of the patch. Eq. 3.1 has two terms: $g(\cdot)$ considers the patch's perceived density after SR, and $h(\cdot)$ accounts for the QoE penalty incurred by SR distortion, quantified by the viewing distance and the EMD (Eq. 2.1) between the upsampled patch and the high-quality patch (ground truth). We empirically define $g(\cdot)$ and $h(\cdot)$ as:

$$g(d_{i,j}, r_{i,j}, \delta_{i,j}) = w_1(\delta_{i,j}) \times d_{i,j} \times r_{i,j} \tag{3.2}$$

$$h(EMD, \delta_{i,j}) = w_2(\delta_{i,j}) \times EMD \tag{3.3}$$

where $w_1(\delta_{i,j})$ and $w_2(\delta_{i,j})$ are weights parameterized on $\delta_{i,j}$. Intuitively, in Eq. 3.2, after SR, the perceived point density improves by a factor of $r_{i,j}$; the QoE gain brought by a higher point density after SR (Eq. 3.2) and the QoE penalty caused by SR distortion (Eq. 3.3) depend on the viewing distance.

Now given a single frame i, we define its quality Q_i as the average of all its visible patches' quality values:

$$Q_i = \frac{\sum_j v_{i,j} q_{i,j}}{\sum_j v_{i,j}} \tag{3.4}$$

where $v_{i,j} \in \{0, 1\}$ is 1 iff the patch is visible, *i.e.,* it falls inside the viewport and is not occluded by other patches. To account for the artifacts caused by patches, we define *inter-patch quality switch* I_i^{patch} as the quality variation across the visible patches within frame i. To account for inter-frame quality switches, we define *inter-frame quality switch* I_i^{frame} as the quality change from frame $i-1$ to frame i:

$$I_i^{patch} = \textbf{StdDev}\left(\{q_{i,j} | \forall j, v_{i,j} > 0\}\right) \tag{3.5}$$

$$I_i^{frame} = \|Q_i - Q_{i-1}\| \tag{3.6}$$

For a volumetric video playback, a possible way to model its overall QoE is a linear combination of Q_i, I_i^{patch}, I_i^{frame}, and I_i^{stall} (the stall of frame i). We choose a linear form that is widely used in 2D Internet videos [248]. Thus, we have

$$QoE = \sum_i Q_i - \sum_i \mu_p(\delta_i) I_i^{patch} - \sum_i \mu_f(\delta_i) I_i^{frame} - \sum_i \mu_s(\delta_i) I_i^{stall} \tag{3.7}$$

Note that depending on the viewing distance, the weights μ_p, μ_f, and μ_s may differ (*e.g.,* viewers may be more sensitive to stalls when watching a scene at a closer distance), so we parameterize the weights with the viewing distance. In Eq. 3.7, δ_i summarizes the viewing distances to all the patches in frame i. We empirically choose $\delta_i = (\sum_j v_{i,j}\delta_{i,j})/(\sum_j v_{i,j})$. Also note that the above

Age	18-25: 21.8%, 26-30: 29.0%, 31-35: 20.4%, 35+: 28.8%
Gender	Male: 60.3%, Female: 39.2%, Other: 0.5%
Country (40 Total)	US: 55.0%, IN: 28.1%, BR: 5.0%, IT: 2.7%, UK: 1.2%, DE: 1.0%, CA: 0.9%, Other: 6.1%
Education	Bachelor: 59.1%, Master: 23.8%, Other: 17.1%

Table 3.1: Demographics of the 1,446 subjects in our user studies.

model is generic and applicable to non-SR-enhanced and non-patch-based volumetric videos as it encompasses special cases without using SR ($r_{i,j}$=1) or patches (I_i^{patch}=0).

3.3.2 Model Validation through User Studies

We next conduct user studies with two purposes: validating our QoE model and deriving the model parameters. Our QoE model considers many factors as described in §3.3.1. The high-level approach of the user study is to let participants subjectively rate the QoE for all the combinations of the above factors' different degrees of impairments, and then use the subjects' ratings to train/validate our QoE model.

We conduct two separate studies referred to as `Study-SR` and `Study-All`. `Study-SR` studies the QoE model for $q_{i,j}$ (Eq. 3.1) while keeping I_i^{patch}, I_i^{frame}, and I_i^{stall} as zero. This allows us to measure the impact of SR without interference from other factors. `Study-All` focuses on the overall QoE model (Eq. 3.7). We obtained IRB approvals for both studies. Instead of performing in-person studies, we conduct both studies online by letting users watch pre-generated videos capturing the rendered viewports (with impairments). We take this approach because: (1) it allows vastly scaling up the study, (2) it helps get diverse users worldwide, and (3) the IRB forbids

Scheme	1×1	1×2	1×3	1×4	2×1	2×2	3×1	4×1
Pt. density	25%	25%	25%	25%	50%	50%	75%	100%
SR ratio	-	×2	×3	×4	-	×2	-	-

Table 3.2: 8 impaired versions (except 4×1) of a video segment. In scheme $m \times n$, m is the point density level and n is SR ratio.

in-person user studies during COVID-19. We have collected responses from 1,446 subjects, whose demographics are shown in Table 3.1.

3.3.2.1 User Study Setup

Volumetric Videos. In our user studies (`Study-SR` in §3.3.2.2 and `Study-All` in §3.3.2.3), we use four videos: *Long Dress* showing a dancing female, *Loot* showing a speaking male, *Band* showing three people playing instruments, and *Haggle* showing three people debating. *Long Dress* and *Loot* are obtained from the 8i dataset [1], each consisting of 800K points per frame for 10 seconds. *Band* and *Haggle* are from the CMU Panoptic dataset [87], each consisting of 300K and 100K points per frame, respectively; we select 10-second segments for our study. For each video, we create 8 versions listed in Table 3.2. Note that since the participants need to watch a large number of impaired copies, the video length (10 seconds) has to be short. Also note that the videos have different point densities, as we want to make the QoE model generic, applicable to different resolutions. We will experimentally verify this shortly.

6DoF Motion Traces and Viewing Distances. We conducted a separate IRB-approved user study (before COVID-19) for collecting 6DoF motion traces of volumetric videos. Specifically, we captured the viewport trajectories of 32 users who watched the four video segments (*Lab*, *Dress*, *Loot*, *Haggle*) introduced in §2.1.1 and above through either a mixed reality headset (Magic Leap One [126]) or an Android smartphone. All these video segments are at their highest quality levels

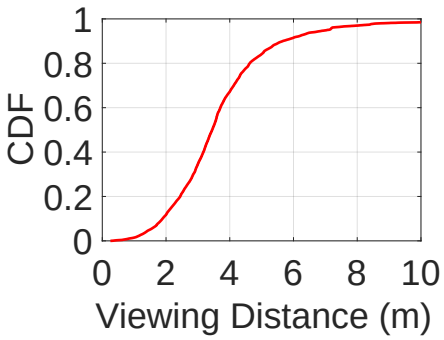


Figure 3.2: Distribution of viewing distance in our motion traces.

(*i.e.,* point densities) without applying SR. We developed custom volumetric video players for both device types. The 6DoF motion data (yaw, pitch, roll, X, Y, Z) was captured at the granularity of 30 Hz. The participants are diverse in terms of their education level (from freshman to Ph.D.), gender (16 females), and age (from 22 to 57). We determine the viewing distances used in our user study (`Study-SR` in §3.3.2.2 and `Study-All` in §3.3.2.3) by analyzing the collected traces. As shown in Figure 3.2, about 70% of the viewing distances are less than 4m. Therefore, we set the maximum viewing distance to be 4m for our user studies, and select the other three distances by evenly dividing this maximum distance into four ranges (*i.e.,* at 1, 2, and 3m).

3.3.2.2 `Study-SR`: How Much QoE Gain Can SR Bring?

We start by studying the QoE gain brought by SR. Specifically, we study the QoE model for $q_{i,j}$ (Eq. 3.1) while keeping I_i^{patch}, I_i^{frame}, and I_i^{stall} as zero. This allows us to measure the impact of SR without interference from other factors.

We use the four videos introduced in §3.3.2.1 for the experiment. We apply our optimized PU-GAN algorithm (details in §3.4.1) to perform upsampling, and create $\binom{8}{2} = 28$ video clips where each clip contains 2 out of 8 versions in Table 3.2 side by side (in a random order). This

approach is known as the double stimulus comparison scale (DSCS) method [85] as recommended by ITU (International Telecommunication Union). We repeat the above process for four viewing distances: 1m, 2m, 3m, and 4m, which are determined from a separate IRB-approved user study whose details are described in 3.3.2.1. To maintain a fixed viewing distance d, we display the viewport at d meters in front of and facing the viewer. We generate 112 video clips at 4K resolution for each video segment.

Next, we design a survey using Qualtrics [167] and publish it on Amazon Mechanical Turk (AMT) [9]. In the survey, we invite each paid AMT subject to view the 112 clips of a random video segment (out of the 4 videos) in a random order. After watching each clip, the subject is asked to rate which side provides a better QoE through 7 choices ("left looks {much better, better, slightly better, similar to, slightly worse, worse, much worse} than right").

Converting User Ratings to Numerical Scores. For a given tuple of (user, viewing distance, video segment), we construct a weighted directed graph for the user based on his/her ratings, where the nodes are the 8 schemes. Assume a video clip contains schemes A (on the left) and B (on the right). If the user thinks that the left (right) is much better, better, or slightly better than the right (left), we add an edge from B to A (A to B) with a weight of 3, 2, and 1, respectively. If the user thinks that the left is similar to the right, we add two edges between A and B, one from A to B and the other from B to A, with both edges' weights set to 0. We then normalize the weights of all the edges to $[0, 1]$ and apply the PageRank algorithm [27] to each graph to compute the weight of every node. We then use the weights (multiplied by 10 for easy interpretation) as the numerical scores of the 8 schemes for the corresponding (user, viewing distance, video segment) tuple. Finally, for each of the 8 schemes under a given viewing distance, we average the numerical scores across all the tuples (of that viewing distance) to obtain the results shown

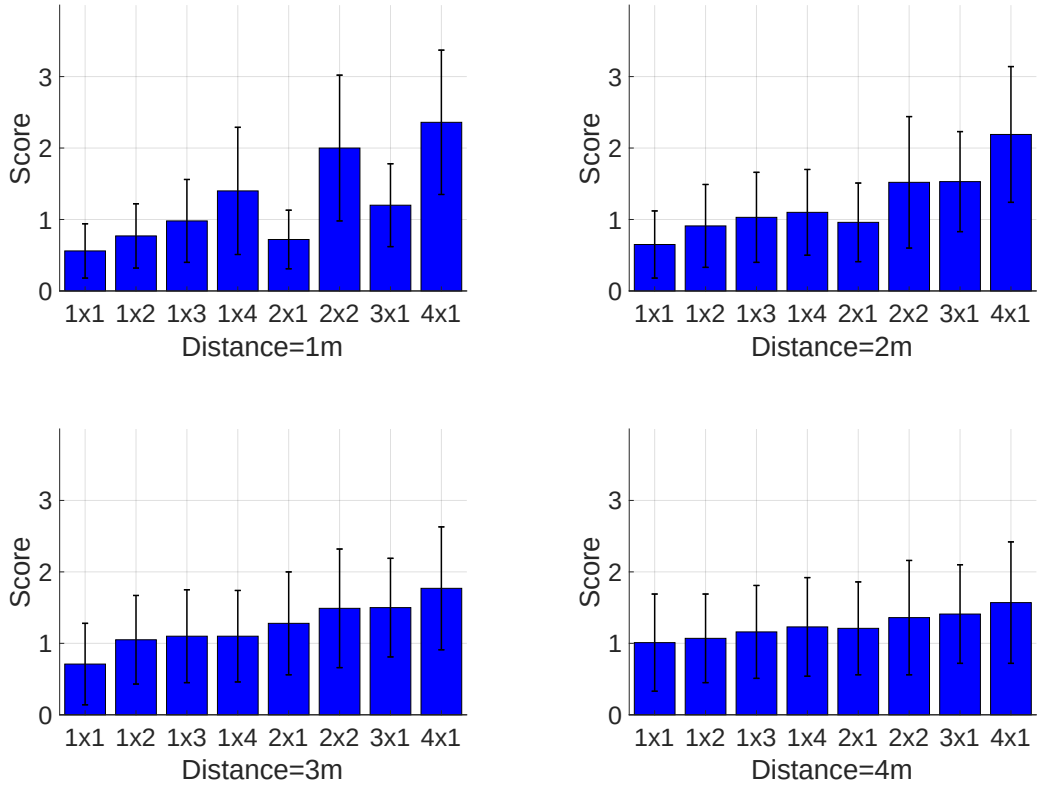


Figure 3.3: The average ratings of the 8 versions across all the users watching all the four video segments (*Long Dress*, *Loot*, *Band*, and *Haggle*).

in Figure 3.3. Note that for each viewing distance, the weights of all the schemes (in each of the graphs) add up to 1. As a result, the numerical scores of the same scheme for different viewing distances are not directly comparable.

Results and Takeaways. We have collected 512 subjects' responses with a total number of 57,344 ratings. Figure 3.3 shows the average ratings of the 8 versions across all the users. The four subplots correspond to the four viewing distances. We make four observations. First, when the viewing distance is short, SR can effectively boost the QoE. For example, at 1m, compared to 1×1, the (user-rated) QoE increases by 37%, 75%, 150% for 1×2, 1×3, and 1×4, respectively; 2×2 improves the QoE by 178% compared to 2×1. Second, under the same point density, the upsampled version's QoE is usually lower than the original content's QoE, in particular when

Videos: {*Long Dress*, *Loot* [1]; *Band*, *Haggle* [87]}
Avg. frame quality Q_i: 7 values uniformly selected from Table 3.2
Avg. distance $dist_{i,j}$: {1m, 2m, 3m, 4m}
Avg. inter-patch switch I_i^{patch}: {0.00,0.45,0.90}
Avg. inter-frame switch I_i^{frame}: {0.00,0.45,0.90}
Avg. stall I_i^{stall}: {0.00,0.01,0.03}

Table 3.3: The factors and their values selected for model validation.

the SR ratio is large. This is caused by SR's distortion. However, the gap tends to reduce as the SR ratio decreases. Third, SR's gain diminishes as the distance increases, because the rendered object becomes smaller in the view. Note that the scores for different distances are not directly comparable. Fourth, the four video segments exhibit similar trends (figure not shown).

3.3.2.3 `Study-All`: How Accurate Is the Overall QoE Model?

Next, we validate the overall QoE model (Eq. 3.7). Similar to `Study-SR` (§3.3.2.2), we use our optimized PU-GAN algorithm (details in §3.4.1) to perform upsampling and create video clips at 4K resolution for four viewing distances: 1m, 2m, 3m, and 4m, which are determined from a separate IRB-approved user study whose details are described in §3.3.2.1. To maintain a fixed viewing distance d, we display the viewport at d meters in front of and facing the viewer. We design a survey using Qualtrics [167] and publish it on Amazon Mechanical Turk (AMT) [9].

We study the impact of all the factors in Eq. 3.7 on the QoE. Table 3.3 lists them and their impairment levels. They lead to a total of 756 combinations for each video segment. Since letting subjects perform $\binom{756}{2}$ pairwise comparisons is infeasible, for each combination, we generate one video clip by putting the impaired version and the high-quality "ground truth" version (4×1, $I_i^{patch} = I_i^{frame} = I_i^{stall} = 0$, same viewing distance) side by side, in a random order. To generate the impaired version, we randomly add perturbations to the patches' quality levels to match the

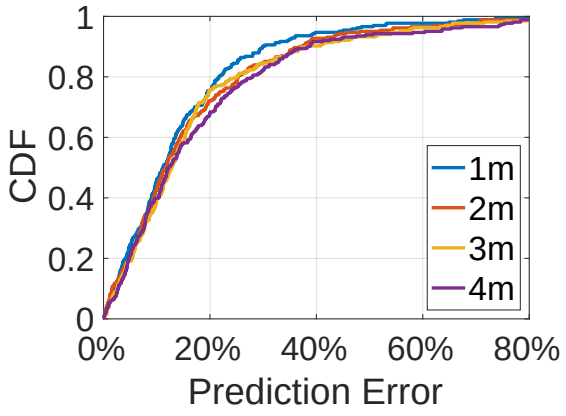


Figure 3.4: QoE prediction error using our model.

corresponding I_i^{patch} and I_i^{frame} values, and randomly inject stalls to match I_i^{stall}. We then ask each subject to watch 100 randomly selected video clips from the 756 clips of a randomly selected video segment. After watching each clip, the subject is asked to rate which side provides a better QoE through 7 choices ("left looks {much better, better, slightly better, similar to, slightly worse, worse, much worse} than right") If the impaired version is {similar to, slightly worse, worse, much worse} than the ground truth, we give the impaired version a score of {3,2,1,0}, respectively.

We have collected 934 subjects' responses with a total number of 93,400 ratings for the above survey published on AMT. For each viewing distance, we use the subjects' ratings to calculate the average score of each of the 756 impaired clips on a scale from 0 to 3, and use it as the QoE ground truth. We then perform 10-fold cross-validation to validate our QoE model (Eq. 3.7, trained using multi-variable linear regression) for each viewing distance. Figure 3.4 plots the CDF of the QoE prediction errors at each viewing distance. The median prediction error for 1m, 2m, 3m, 4m is 11.4%, 12.2%, 12.8%, and 12.9%, respectively. The (Person, Spearman) correlation coefficients between the ground-truth QoE score and the predicted QoE score are also high: (0.89, 0.89) at 1m, (0.87, 0.88) at 2m, (0.87, 0.88) at 3m, and (0.85, 0.85) at 4m.

δ	*D*: *Long Dress*; *L*: *Loot*; *B*:*Band*; *H*:*Haggle*			
	$DBH \Rightarrow L$	$LBH \Rightarrow D$	$DLB \Rightarrow H$	$DLH \Rightarrow B$
1m	0.80	0.74	0.86	0.85
2m	0.76	0.71	0.87	0.87
3m	0.80	0.73	0.83	0.87
4m	0.78	0.71	0.76	0.80

Table 3.4: Spearman correlation coefficient between QoE ground truth and cross-video prediction. $XYZ \Rightarrow W$ means using the model trained from videos X, Y, and Z to predict video W's QoE.

δ	*Long Dress* + *Loot* + *Band* + *Haggle*				
	w_1	w_2	μ_p	μ_f	μ_s
1m	0.55	27.80	0.52	0.40	170.5
2m	0.42	39.83	1.05	0.91	149.8
3m	0.27	26.63	1.23	1.04	176.7
4m	0.16	17.17	0.47	0.06	304.1

Table 3.5: Parameters of the final model used in YuZu.

The above QoE models are trained from all four videos. Table 3.4 shows the Spearman correlation coefficients between the ground-truth QoE and *cross-video* prediction results. We use the data of three videos to train a QoE model and use it to predict the QoE for the remaining video. The results indicate that the same QoE model and its parameters are applicable to volumetric content of the same genre (portraits of people – a major application of volumetric streaming – in our case). We also confirm that most parameters trained from different video segments are indeed quite similar, in spite of the segments' different point densities. When applied to other genres, the model's parameters may differ, as to be explored in our future work (the same happens to 2D videos [247]). Table 3.5 lists our final model's parameters trained using the entire dataset. The model will be used by YuZu.

3.4 System Design of YuZu

We now detail the system design of YuZu (Figure 3.1) that addresses the challenges we identified in §2.1.1.

3.4.1 Accelerating SR Upsampling

To accelerate 3D upsampling, we take a principled approach by exploring three orthogonal directions:

• **Model Optimization.** How to simplify the upsampling logic while retaining the inference accuracy? (§3.4.1.1)

• **Data Reduction.** How to strategically feed less data to SR models with negligible impact on QoE? (§3.4.2)

• **Pre-processing and Post-processing Trimming.** How to simplify the sophisticated pre- and post-processing stages without incurring side effects on inferences? (§3.4.1.2)

Our optimizations can apply to all 3D SR models we have investigated [106, 229, 234, 249] and they are video-agnostic. In §3.6, we demonstrate the optimization results for two SR models: PU-GAN [106] and MPU [229].

3.4.1.1 SR Model Optimization

We take a "top-down" approach by first optimizing the model as a whole and then fine-tuning its detailed structure. For most machine learning models (including 2D SR), after performing an inference, the input is no longer needed and will be discarded. Our investigated 3D SR models [106, 229, 234, 249] make no exception. We instead make a fundamental observation regarding 3D

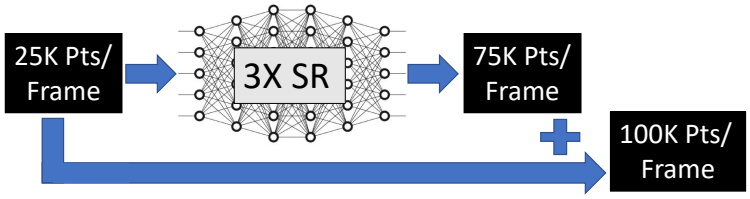


Figure 3.5: Using a 3× SR model to realize 4× SR.

point clouds. Different from a 2D image, a point cloud is a set of unstructured points, which means that point clouds can be *merged via a simple set union operation*. We also note that 3D SR's output points *refine and differ from* the input. Based on this key insight, we propose a simple yet effective optimization: YuZu merges the input low-density point cloud with the SR output in order to improve the visual quality, or to reduce the computation overhead while maintaining the same upsampling ratio. For example, as shown in Figure 3.5, to achieve 4× upsampling, instead of using a 4× SR model, we can use a (computationally more efficient) 3× SR model and merge the input with the output. Since SR exploits the overfitting nature of DNN, the spatial distributions of upsampled points and the ground truth are expected to be highly similar. By leveraging the input data and downgrading the SR ratio from 4× to 3×, we can achieve an acceleration of up to ~35% without hurting the SR accuracy (Figure 3.9). Note that in offline training, the loss function is computed *after* merging the input low-density point cloud with the SR output. This makes the trained models aware of and adaptive to the merging process, improving the upsampling accuracy compared to computing the loss function before that.

Next, we explore modifying 3D SR model's DNN structure for inference acceleration. By profiling the inference time of PU-GAN, we find that its three stages, feature extraction, feature expansion, and point set generation, take 78.3%, 19.3%, and 2.4% of execution time, respectively (4× SR). Within the feature extraction stage that dominates the runtime overhead, most operations

are *convolutions*. We make the same observation for other 3D SR models that we investigated [229, 234, 249].

To accelerate convolutions, we replace the original feature extraction, which (*e.g.,* in the case of PU-GAN) enhances the solution in PointNet++ [164] through dynamic graph convolution [190], with a recent proposal called spherical kernel function (SKF) [105]. SKF partitions a 3D space into multiple volumetric bins and specifies a learnable parameter to convolve the points in each bin. In contrast to continuous filter approaches (*e.g.,* multilayer perceptron) used in existing SR models, SKF is a *discrete* metric-based spherical convolutional kernel, and is thus computationally attractive for dense point clouds. Moreover, it is applicable to all the 3D SR models we examined. We find that SKF brings no degradation to the upsampling accuracy (§3.6.3). One reason may be that the kernel asymmetry of SKF facilitates learning fine geometric details of point clouds [105].

In addition to utilizing SKF, we conduct layer-by-layer profiling [245, 42] to fine-tune the SR model's performance-accuracy tradeoff. Take PU-GAN as an example. We remove the last two dense layers of feature extraction and several heavyweight convolution layers in the feature expansion stage, as they make limited contributions to the upsampling accuracy. We also judiciously remove a small number of expanded features to reduce the GPU memory footprint. For other 3D SR models, their model tuning follows a similar approach.

3.4.1.2 Trimming Pre- and Post-Processing

Recall from §2.1.1 that to ensure a manageable model complexity, a 3D SR model divides a point cloud into small patches as basic units for upsampling. We discover that as an important preprocessing step, the patch generation process incurs a high overhead. For example, PU-GAN

generates the patches by applying kNN to the seeds created by downsampling. Since the generated patches may overlap, after upsampling, PU-GAN needs to perform post-processing: it applies the furthest point sampling [137] to remove duplicated points.

To mitigate the above overhead, YuZu adopts a simple patch generation method. It divides the space into cubic cells, and assigns each non-empty cell (*i.e.,* a cell that contains points) to a patch. Compared to the default patch generation approaches used by PU-GAN and other 3D SR frameworks [229, 106], our approach runs very fast; it also brings no overlap among patches, thus eliminating the post-processing step (*i.e.,* overlap removal). In addition, the patches now have a simple geometry shape, so that they can be indexed, searched, and manipulated at runtime. Meanwhile, We find that our patch generation approach does not sacrifice the upsampling accuracy and may even improve the accuracy compared to vanilla PU-GAN and MPU (§3.6.3). This is likely because cubic cells provide a more consistent structure for the patches, making it easier to perform SR. We also investigate several other patch generation methods based on Voronoi diagram [52] and 3D SIFT [187], but none outperforms our cubic-cell-based approach from either the performance or the accuracy perspective.

3.4.2 Caching and Reusing SR Results

Videos usually exhibit similarities across frames. We find that volumetric videos make no exceptions. This indicates rich opportunities for caching and reusing SR results.

At a high level, YuZu reuses SR results based on the similarity between patches, which is the basis of inter-frame encoding. Inter-frame similarity has been extensively studied and exploited in 2D videos. However, none of the 2D inter-frame encoding techniques can be directly

applied to volumetric videos due to the fundamental difference between pixel-based 2D frames and volumetric frames consisting of unstructured points. There are very few studies on 3D inter-frame encoding [88, 132]; they are incompatible with YuZu's patch-based upsampling, and incur high complexity hindering line-rate decoding. Due to the above reasons, we design our own SR caching/reusing algorithm. Our algorithm is agnostic of and orthogonal to a specific SR model.

YuZu reuses 3D SR results on a per-patch basis to match the patch-based upsampling procedure. Recall from §3.4.1.2 that YuZu generates patches using 3D cubic cells. Let $p(i,j)$ denote patch j of frame i, and let $N(i,j)$ denote the number of points in $p(i,j)$. YuZu allows reusing the SR result of $p(i,j)$ for subsequent *consecutive patches at the same location*, *i.e.,* $p(i{+}1,j), p(i{+}2,j)$, and so on. YuZu restricts reusing patches only at the same location due to two considerations. First, we empirically observe that most patch similarities indeed occur at the same cell location; this makes the benefits (in terms of reduced SR overhead) of reusing a patch belonging to a different cell marginal. Second, allowing reusing a patch at a different cell will drastically increase the overhead of pre-computing the caching/reusing decisions.

We now describe YuZu's SR reuse algorithm. YuZu first determines offline the similarity of two patches. For each patch pair $(p(i,j), p(i+1,j))$, YuZu computes a Weighted Complete Bipartite Graph [12] $B : p(i,j) \rightarrow p(i+1,j)$, which we find to be suitable for dealing with unstructured points. In the bipartite graph, there is a directed edge from every point in $p(i,j)$ to every point in $p(i+1,j)$, and the weight of the edge is their Euclidean distance. We then calculate the minimum-weight matching (MWM) [205] for the graph, *i.e.,* finding $N(i,j)$ edges such that (1) these edges share no common vertices (points), and (2) the sum of their weights is minimized. Intuitively, the MWM identifies a transformation from $p(i,j)$ to $p(i+1,j)$ with a minimum moving distance for the points. The Hungarian algorithm [12] that computes the

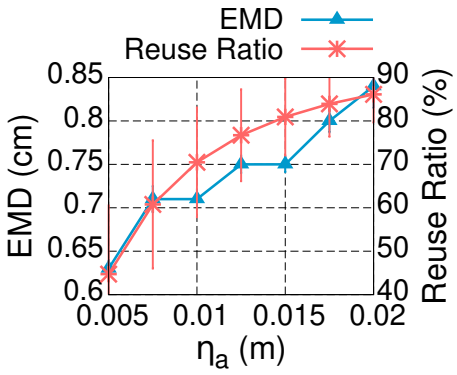


Figure 3.6: Impact of η_a (using the video in §2.1.1, 1×4 SR).

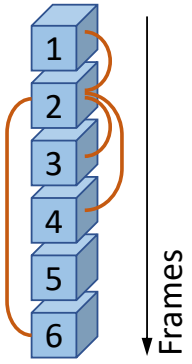


Figure 3.7: Reusing SR results across consecutive frames.

MWM has a complexity of $O(N^4)$ where $N = \max\{N(i,j), N(i+1,j)\}$. We instead employ a faster $O(N^2)$ approximation algorithm that is found to work well in practice.[‡]

We call every edge in the MWM a point motion vector (PMV). A PMV differs from a 2D video's motion vector, which represents a macroblock in a frame based on the position of the same or a similar macroblock in another reference frame. Leveraging the PMVs, we determine that $p(i+1,j)$ and $p(i,j)$ are *similar* if three criteria are satisfied. (1) $N(i,j)$ and $N(i+1,j)$ differ by no more than $\eta_n\%$; (2) the average length of all the PMVs is smaller than η_a; (3) the top 90-percentile of the shortest PMV is smaller than η_v. These three criteria dictate that $p(i,j)$ and $p(i+1,j)$ have a similar number of points, and the points' collective motions are small. Figure 3.6 shows how η_a impacts EMD and the patch reuse ratio (% of patches that can reuse a previous SR result). As shown, increasing η_a increases the reuse ratio, but meanwhile decreases the accuracy. According to Figure 3.6, we set η_a to 0.01m to balance the performance and accuracy. Using similar methods, we empirically set η_n=10 and η_v=0.01m.

[‡]The approximation algorithm sorts all the edges by their weights in ascending order. It then adds the edges to the MWM in that order and skips edges that share points with an existing edge in the matching, until every point in $p(i,j)$ or every point in $p(i+1,j)$ is in the MWM.

Next, we consider how to reuse an SR result across multiple patches belonging to consecutive frames. We define $sim_j(i_1, i_2) \in \{0, 1\}$ to be 1 if and only if $p(i_1, j)$ and $p(i_2, j)$ are similar, *i.e.,* satisfying the above three criteria where $i_2 > i_1$. Figure 3.7 shows an example of 6 consecutive patches at location j where $\forall 1 \leq x < y \leq 6 : sim_j(x, y) = 0$ except that $sim_j(1, 2)$, $sim_j(2, 3)$, $sim_j(2, 4)$, and $sim_j(2, 6)$ are 1. YuZu allows a patch's SR result to be reused across *consecutive* patches if they are all similar to the first patch. For example, Patches 3 and 4 can reuse Patch 2's SR result. However, YuZu does not let Patch 6 reuse Patch 2 because $sim_j(2, 5) = 0$. We make this design decision for two reasons. First, we observe that non-consecutive patches are unlikely to be similar in real volumetric videos. Second, supporting non-consecutive reuse requires computing $sim_j(x, y) \forall x < y$, making offline video processing slow.

We develop an algorithm that *minimizes the number of patches to be upsampled*, to boost the online SR performance. For example, in Figure 3.7, the minimum number of patches to be upsampled is 4: Patches 1, 2, 5, and 6. YuZu efficiently and optimally solves this through dynamic programming (DP). Given n patches $p(1, j), ..., p(n, j)$ and their sim_j information, let $u(i, j)$ be the minimum number of patches that need to be upsampled in $\{p(i, j), ..., p(n, j)\}$ if we decide to upsample $p(i, j)$. Then $u(i, j)$ can be derived through DP as:

$$u(i, j) = \min \left\{ u(i + 1, j), \min_{i<k\leq n:\forall i<t\leq k:sim_j(i,t)=1} \{u(k + 1, j)\} \right\} + 1 \tag{3.8}$$

The RHS of Eq. 3.8 examines each patch following $p(i, j)$ and updates $u(i, j)$ if stopping reusing $p(i, j)$ at $p(k + 1, j)$ yields a better $u(i, j)$. The search continues until hitting a patch that is not similar to $p(i, j)$. Eq. 3.8 can be solved backwards starting from $u(n + 1, j) = 0$. The solution is $u(1, j)$.

Since YuZu streams VoD volumetric content, all the above logic (calculating MWM, sim_j, and DP) is performed *offline* for each patch location j. Thus, there is no runtime overhead. The SR reuse decisions are sent to the client as meta data, which is only 0.5KB per frame for our testing video in §2.1.1.

3.4.3 Network/Compute Resource Adaptation

YuZu adapts to not only the fluctuating network condition (similar to the job of traditional bitrate adaptation algorithms [128, 243, 248]), but also the available compute resource, due to the high computation overhead of 3D SR. More importantly, these two dimensions incur a tradeoff: given a fixed playback deadline, should YuZu download high-resolution content, or download lower-resolution content and spend time upsampling it? Fortunately, our QoE model (§3.3.1) dictates how to quantitatively balance this tradeoff.

We first formulate an online network/compute adaptation problem. The video is divided into n chunks each consisting of f frames. To achieve fine-grained adaptation, each chunk is further spatially segmented into b *blocks* (*e.g.,* b=5^3), which are the atomic scheduling units in YuZu's adaptation algorithm. Each block consists of multiple patches (recall from §3.4.1.2 that each patch occupies a cubic cell). At runtime, YuZu considers all the blocks belonging to a finite horizon of the next w chunks, and searches for their quality and SR ratio assignments that maximize the QoE defined in Eq. 3.7. This formulation extends the model predictive control (MPC) scheme [248] that proves to be effective for traditional 2D video streaming. The solution space is $O(8^{wb})$ (the 8 possible assignments are listed in Table 3.2).

We consider how to efficiently solve the above discrete optimization problem. An exhaustive search is clearly infeasible. Due to the large solution space, even the memorization approach (FastMPC [248]) is not practical. Another possibility is a learning-based approach such as Pensieve [128]. However, it requires offline training and may incur a non-trivial inference overhead. Moreover, a recent work [243] indicates that reinforcement learning based bitrate adaptation solutions do not necessarily outperform simple buffer-based approaches [81].

To overcome the above challenges, we develop a lightweight approximation algorithm. It executes in two stages: first determine the quality and SR ratios of to-be-downloaded chunks, and then fine-tune the SR ratios before upsampling. Specifically, in the first stage, *before downloading each chunk*, YuZu performs a *coarse-grained search* by assuming that all the blocks in each chunk have the same quality/SR-ratio assignment. The rationale is that, at this moment, the playback deadline is still far away (compared to Stage 2), and thus the network/computation-load uncertainty diminishes the benefits brought by a block-level, fine-grained search. Meanwhile, this reduces the solution space from $O(8^{wb})$ to $O(8^{w})$. Specifically, we (1) start with a quasi-optimal solution obtained from an even coarser-grained search at the granularity of every two consecutive chunks, and (2) perform pruning by bounding [19]. After the above two optimizations, for a practical w (*e.g.,* w=10), the search time (for maximizing the QoE in Eq. 3.7) becomes negligible compared to the downloading and upsampling time. To estimate I_i^{stall} in Eq. 3.7, at runtime, YuZu continuously estimates (1) the network bandwidth using the method in [73] and (2) the local processing time of a frame using EWMA-based estimation.

The second stage takes place *before upsampling each frame*. At this stage, the playback deadline gets closer and thus a *block-level, fine-grained search* would be beneficial. To reduce the search complexity, YuZu employs Simulated Annealing (SA) [94] – a probabilistic, greedy approach that

approximates the global optimum. Our algorithm begins with setting all the blocks' SR ratios to the lowest (no SR). For each block, the algorithm tries to increase its SR ratio by one level. If the resulting QoE of the finite horizon increases, this change is always accepted; otherwise, we may still accept this change with a probability of $exp(-\frac{\Delta}{t})$, where Δ is the decrease of the QoE and t is the current number of iterations, to avoid a potential local maximum. To speed up the SA algorithm, we reduce the finite horizon to two frames: the previous frame and the current (to be upsampled) frame – we empirically find that conducting frequent adaptations with a short horizon at a per-frame basis outperforms infrequent adaptations with a long horizon at a per-chunk basis in terms of the QoE.

3.4.4 Coloring SR Results

As described in §2.1.1, none of the 3D SR models we investigated performs colorization. There are two high-level approaches for colorization. One is augmenting the SR models by adding the color component. This may yield good colorization results, but at the cost of significantly increasing the SR workload. Given this concern, YuZu takes a much more lightweight approach: approximating each upsampled point's color using the color of the nearest point in the low-density point cloud (*i.e.,* the input to the SR model).

In particular, YuZu employs two mechanisms to speed up the nearest point search. First, the search is performed on an octree [207], which recursively divides a point cloud (as the root node) into eight octants, each associated with a child node. The levels of detail of the point cloud are controlled by the height of the tree. Performing nearest point search on an octree has a low complexity of $O(logN)$ where N is the number of nodes in the tree.

Second, YuZu caches and reuses the results of previously searched points. The cache is indexed by a point's discretized coordinates, and the cached value is the color looked up from the octree. When coloring an upsampled point, YuZu first performs cache lookup in $O(1)$; upon a hit, the cached color will be directly used as the color of the point; otherwise, YuZu performs a full octree search and adds the search result to the cache. The discretization granularity incurs a tradeoff between colorization performance and quality. We empirically observe that a discretization granularity of 1cm^3 can yield good visual quality under typical viewing distances ($\geq$ 1m).

We also notice opportunities for further improving the colorization quality. For example, the nearest point approach can be generalized into interpolating the nearest k points' colors; it can also be used in conjunction with DNN-based colorization, which may be more suitable for patches with complex, heterogeneous colors. Nevertheless, these enhancements inevitably increase the runtime overhead. We will explore them in future work.

3.5 Implementation

We integrate all the components in §3.4 into YuZu, a holistic system as shown in Figure 3.1. Our implementation consists of 10,848 lines of code (LoC), with 8,326 LoC for the client.

For offline SR model training, we modify the source code of PU-GAN [162] and MPU [138] using TensorFlow 1.14 [206] and custom TensorFlow operators from SPH3D-GCN [198]. Our pre-trained models are saved in the ProtoBuf format [161] that is language- and platform-neutral, facilitating future reuse. For online streaming, we implement the client player on Linux in C++. We use the Draco Library [46] for encoding and decoding the point cloud data. We employ Bazel [22] to compile the TensorFlow 1.14 C/C++ library and use the compiled library to load

and execute the SR models. The client *pipelines* content fetching (network-bound), point cloud decoding & patch generation (CPU-bound), 3D SR (GPU-bound), and colorization (CPU-bound) of different frames for better performance. The server is also built in C++, with a custom DASH-like protocol over TCP for client-server communication.

3.6 Evaluation

3.6.1 Experimental Setup

Volumetric Videos. We use four point-cloud-based volumetric videos throughout our evaluations. (1) Our own video. We capture a volumetric video by ourselves using 3 synchronized depth cameras. It has 3,622 frames (2 min) each consisting of ~100K points. We refer to this video as *Lab*. We have used it to motivate YuZu in §2.1.1. (2) The Long Dress (*Dress*) and *Loot* videos (§3.3.2). They have 300 frames (10 sec) each consisting of ~100K points. Since they are short, we loop them (with cold caches) 10 times in our evaluations. (3) The *Haggle* video (§3.3.2). It has 7,800 frames (4'20") each consisting of ~100K points. For all four videos, the eight possible resolution/SR-ratio assignments are listed in Table 3.2. For each video, we train their SR models separately. All the videos are at 30 FPS, encoded by Draco [46]. Unless otherwise mentioned, the results reported in the remainder of this section are generated using all four videos. The average encoded bitrate of *Lab*, *Dress*, *Loot*, and *Haggle* (4×1) are 96, 108, 118, and 118 Mbps, respectively.

3D SR Models. We apply our developed model acceleration techniques to two recently proposed 3D SR models: PU-GAN [106] and MPU [229]. The two models usually yield qualitatively similar results, so we show the results of PU-GAN by default. For certain SR-specific experiments

(*e.g.,* SR acceleration), we show both models' results. The models are trained on a per-video basis. For each video, the total size of all its models (×2, ×3, and ×4) is around 1.25 MB.

Metrics and Roadmap. We thoroughly evaluate YuZu in terms of performance, QoE, and resource utilization. §3.6.2 evaluates the QoE improvement brought by our 3D SR optimizations using both subjective (*i.e.,* real-user ratings) and objective (*e.g.,* PSNR [76]) metrics. §3.6.3 focuses on the performance gain of our 3D SR optimizations, from the perspectives of resource usage, inference time, and upsampling accuracy. §3.6.4 and §3.6.5 evaluate the end-to-end performance (*e.g.,* QoE and data usage) of YuZu. §3.6.6 provides additional micro benchmarks.

Network Conditions. We consider the following network conditions that are readily available in today's wired and wireless networks. (1) *Wired network with stable bandwidth* (*e.g.,* 50, 75, and 100 Mbps) and ~10ms RTT. (2) *Fluctuating bandwidth* captured from real LTE networks. We collect 12 bandwidth traces from a major LTE carrier in multiple U.S. states at diverse locations (campus, malls, streets, *etc.*). Across the traces, their average bandwidth varies from 33.7 to 176.5 Mbps, and the standard deviation ranges from 13.5 to 26.8 Mbps. We use `tc` [115] to replay these traces (with a 50ms base RTT typically observed in LTE [90]). (3) We also conduct *live LTE experiments* at 9 diverse locations in a U.S. city where the average bandwidth varies from 41.1 to 52.4 Mbps and the standard deviation is between 16.6 and 20.7 Mbps.

Devices. We use a commodity machine with an Intel Core i7-9800X CPU @ 3.80GHz and 32GB memory as the YuZu server. We use three client hosts: (1) a desktop with an Intel Core i9-10900X CPU @ 3.70GHz, an NVIDIA GeForce RTX 2080Ti GPU, and 32GB memory (the default client used in our evaluations); (2) a desktop with the same CPU, an NVIDIA GeForce GTX

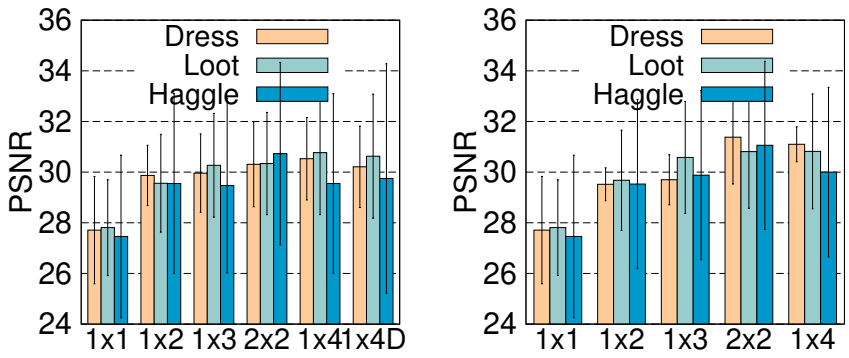


Figure 3.8: PSNR of YuZu (left) and vanilla PU-GAN (right).

1660Ti GPU, and 32GB memory; (3) an NVIDIA Jetson TX2 embedded system board with a Pascal-architecture GPU of 256 CUDA Cores, 8GB memory, and a quad-core CPU. They represent a typical high-end PC, a medium-class PC, and a mobile device, respectively.

User Motion Traces. We collect 32 users' 6DoF motion traces when watching the four videos, and replay them in some experiments. The details about how we collect the motion traces can be found in §3.3.2.1.

3.6.2 SR Quality

Subjective Ratings. Recall that in our user studies, we ask our participants to rate the SR results generated by our optimized SR scheme (§3.4.1). Figure 3.3 shows that SR brings a significant boost to the user-perceived QoE. For example, at 1m, compared to 1×1, the user-rated QoE increases by 37%, 75%, and 150% for 1×2, 1×3, and 1×4, respectively; 2×2 improves the QoE by 178% compared to 2×1 (§3.3.2).

Objective Metric. We also examine how SR improves PSNR [76], an objective metric of image quality. The methodology is as follows. We replay the 32 users' 6DoF motion traces of watching the videos under different SR settings, and save the rendered viewports as images $\{I_{SR}\}$. We then

repeat the above process using the original videos (4×1), and capture the viewport images $\{I_{4\times1}\}$. We compute the PSNR values by comparing each image in $\{I_{SR}\}$ with its corresponding image in $\{I_{4\times1}\}$. Figure 3.8 (left) shows the PSNR values for 1×1, 1×2, 1×3, 2×2, 1×4, and 1×4 with reusing SR results (denoted as "1×4D") across all the captured viewports. We notice a significant increase of PSNR from 1×1 to 1×2. The PSNR also increases marginally from 1×2 to 1×4. Meanwhile, the PSNR change between 1×4 and 1×4D is negligible, indicating that caching and reusing SR results brings little impact on the perceived video quality (but drastic performance gain as shown in §3.6.3). The results of *Lab* are similar. Note that a PSNR value over 30 typically indicates good visual quality [42, 208]. Figure 3.8 (right) shows the PSNR values for the unmodified PU-GAN model. The qualitatively similar results between the left and right plots of Figure 3.8 indicate that our SR acceleration modifications sacrifice little visual quality. Note the above results include the colorization step, which is described in §3.4.4 and separately evaluated in §3.6.6.

Comparing Figure 3.3 and Figure 3.8, we notice disparities between users' QoE ratings and PSNR values. This indicates that image qualities of rendered 2D content do not directly reflect the perceived QoE of volumetric content. This is a key reason for developing the QoE model for volumetric videos.

3.6.3 SR Performance Breakdown

We now take a closer look at the effectiveness of each of our proposed methods for accelerating SR. As listed in Table 3.6, M_1 denotes the vanilla 3D SR model as the comparison baseline; M_2 to M_6 are our proposed SR acceleration methods in §3.4.1 and §3.4.2. They are presented in a *cumulative* fashion, *i.e.,* M_i includes every feature of M_{i-1} plus some new feature. The experiments are

M_1	The vanilla 3D SR model (PU-GAN and MPU)
M_2	M_1 and optimizing patch generation
M_3	M_2 and layer profiling & pruning
M_4	M_3 and applying the spherical kernal function (SKF)
M_5	M_4 and merging SR input with SR output
M_6	M_5 and caching/reusing SR results

Table 3.6: SR acceleration methods (cumulative).

conducted using two 3D SR models (PU-GAN [106] and MPU [229]), 100Mbps wired network, 4× SR, with network/compute resource adaptation (§3.4.3) disabled.

Figures 3.9 and 3.10 show the results of PU-GAN and MPU on the PC (2080Ti) and Jetson TX2 board, respectively. On the Jetson board, due to its low compute power (and mobile devices' small screen size), we reduce the original video's resolution from 100K to 20K points per frame (*i.e.,* the SR is from 5K to 20K points per frame). We consider four metrics: (1) maximum GPU memory usage (on Jetson TX2 we measure the system memory shared by GPU and CPU), (2) average upsampling speed (in FPS), (3) inference accuracy measured in EMD between each upsampled frame and the ground truth (4×1), and (4) visual consistency measured in EMD between each consecutive pair of upsampled frames.

As shown, on 2080Ti, for PU-GAN (MPU), compared to M_1, M_6 reduces the GPU memory usage by 87% (90%), accelerates the upsampling by 307× (542×), improves the average upsampling accuracy by 24% (14%), and slightly improves the consistency. Also, each optimization (M_2 to M_6) individually improves the upsampling speed and possibly other metric(s). The Jetson setup shows a similar trend. The two models (PU-GAN and MPU) we studied exhibit similar performance gains as we progressively apply our optimizations, except that MPU is less sensitive to M_5. This is because of the network structure difference between PU-GAN and MPU. Note that we do not

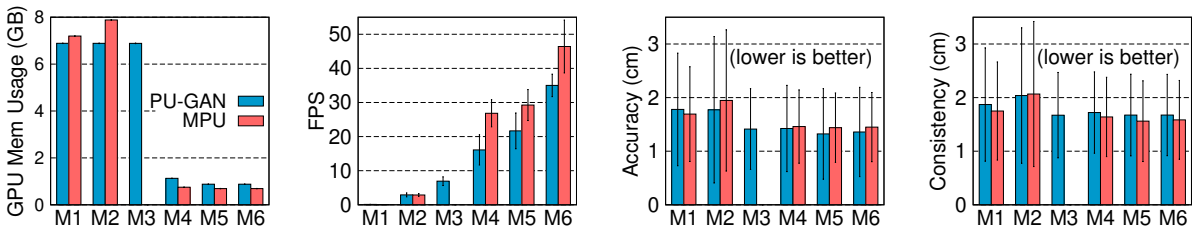


Figure 3.9: Memory usage, upsampling FPS, upsampling accuracy, and visual consistency of M_1 to M_6 (2080Ti desktop).

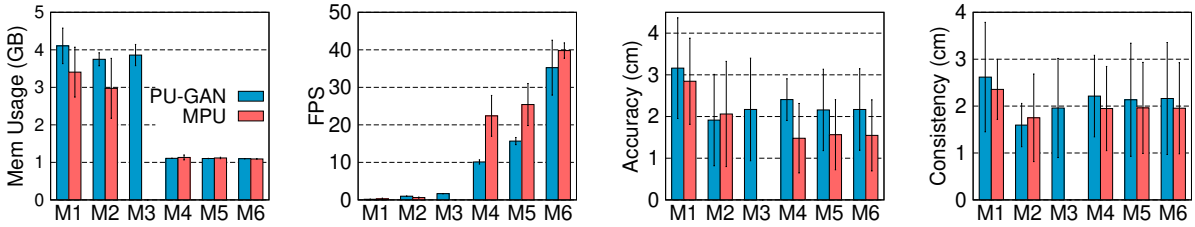


Figure 3.10: Memory usage, upsampling FPS, upsampling accuracy, and visual consistency of M_1 to M_6 (Jetson TX2 board).

apply M3 to MPU because our layer-by-layer profiling (§3.4.1.1) reveals there is no layer that only makes a marginal contribution to the overall upsampling accuracy in the MPU model.

Latency Breakdown. Figure 3.11 shows the latency breakdown of processing an average frame using PU-GAN (*Lab* video, wired 100Mbps, 2080Ti desktop) under two settings: 2×2 and 1×4. As shown, SR remains the most time-consuming component. The breakdown for MPU is similar. The above results indicate the importance of SR acceleration.

3.6.4 Diverse Network Conditions

We evaluate the QoE of YuZu under different network conditions, using the four videos and the associated motion traces.

Stable Bandwidth. We first consider two stable bandwidth: 50Mbps and 75Mbps. Under each bandwidth profile, we run the full-fledged YuZu ("Full") and six statically configured YuZu

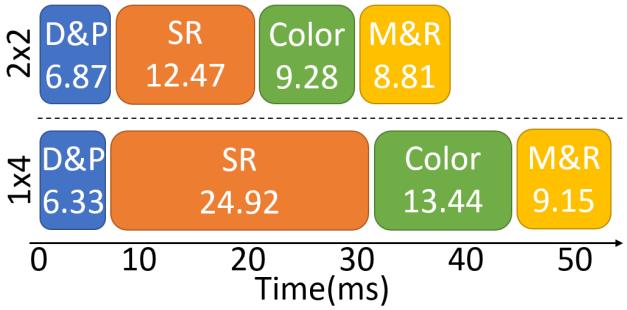


Figure 3.11: Frame processing time breakdown. D&P: decoding and patch generation; SR: upsampling; Color: colorization, M&R: merging and rendering.

instances: 4×1, 2×2, and 1×4 with and without SR result reusing. The QoE results are shown in Figure 3.12. We make several observations. First, when the bandwidth is low (50Mbps), 4×1 (without SR) gives the lowest (and even negative) QoE. This is because the limited bandwidth leads to high *network-incurred* stall when fetching high-resolution content; SR can effectively improve the QoE by using computation to compensate for the low bandwidth. Second, when the bandwidth increases to 75Mbps, 1×4 gives the lowest QoE due to the distortion and *computation-incurred* stall due to the high SR ratio. Instead, when the bandwidth is sufficient, the player should fetch the content with a higher quality (*e.g.,* 4×1D). Third, caching and reusing (C&R) the SR results improves the QoE when either the bandwidth is low (*e.g.,* 4×1 at 50Mbps), or the SR ratio is high (*e.g.,* 1×4). Under these two scenarios, C&R reduces the network and compute resource usage, respectively. The saved resources can be used to improve the content quality for other frames with more heterogeneity.

Figure 3.13 compares the (normalized) data usage, which is defined as the total downloaded bytes including the SR models and meta data. Compared to 4×1, applying C&R reduces the data usage by 40.5%. Also, increasing the SR ratio reduces the data usage, *e.g.,* 1×4D consumes only 18.3% of the data compared to 4×1. The full-fledged YuZu with adaptation gives the overall best

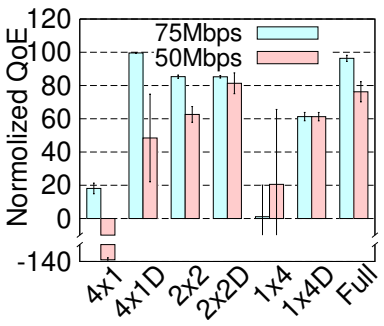


Figure 3.12: QoE over stable bandwidth ("D"=caching & reusing SR results).

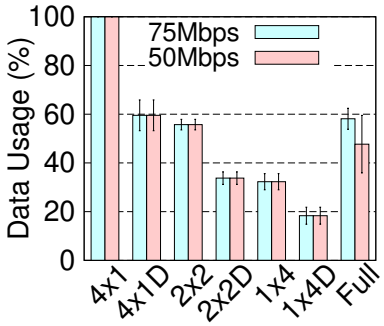


Figure 3.13: Data usage over stable bandwidth.

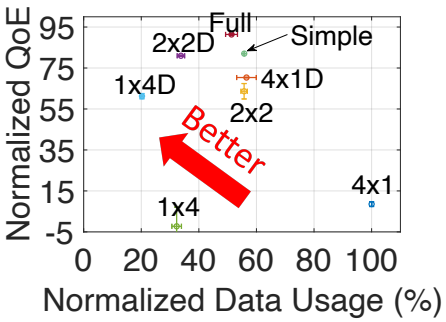


Figure 3.14: QoE vs. Data usage over fluctuating bandwidth (LTE traces).

QoE (Figure 3.12) and low data usage (Figure 3.13) by balancing the compute and network resource consumption. Compared to 4×1, full YuZu reduces the data usage by 52.3% (50Mbps) and 41.9% (75Mbps) while boosting the QoE by 214% (50Mbps) and 78.3% (75Mbps).

Fluctuating Bandwidth. We repeat the above experiment over fluctuating bandwidth emulated using our collected LTE traces (§3.6.1). The results are shown in Figure 3.14, which considers both the data usage (x-axis) and the QoE (y-axis). 4×1 yields the highest data usage; further applying C&R (4×1D) not only reduces the data usage by 40.5%, but also increases the QoE by 61.8% due to reduced stall. The full YuZu further improves the QoE by 21.0% and reduces the average data usage by 8.2%. This is achieved through strategically fetching lower-quality blocks and using higher SR ratios. In addition, the full YuZu improves the QoE by 10.4% to 93.7%, compared to 1×4 and 2×2 with and without C&R.

Live LTE. We conduct live LTE experiments at 9 locations in a major U.S. city. As shown in Figure 3.15, the results are largely aligned with those in Figure 3.14, except for the lower QoE of 4×1. This is because of the lower bandwidth of live LTE throughout the test locations compared

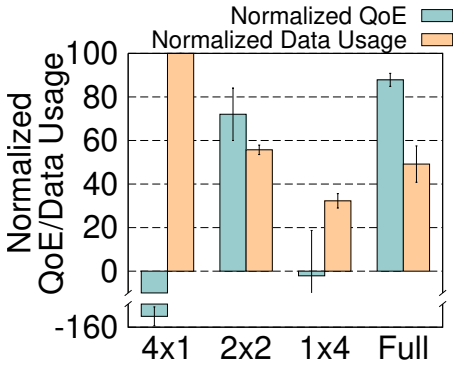


Figure 3.15: QoE and data usage over live LTE networks.

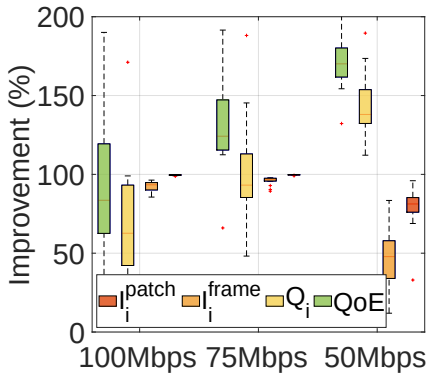


Figure 3.16: YuZu over ViVo.

to the LTE traces used in Figure 3.14. Compared to 4×1, the full YuZu improves the QoE by 210.3% and reduces the data usage by 50.8%.

3.6.5 YuZu vs. Existing Approaches

YuZu vs. Viewport-Adaptive Streaming. We compare YuZu with ViVo [68], a recently proposed viewport-adaptive approach. Leveraging 6DoF motion prediction, ViVo determines what content to fetch and which quality to fetch based on predicted viewport and viewing distance. Similar viewport-adaptive approaches are used in the other systems [102, 156].

We develop a custom replication of ViVo on Linux in 7,101 LoC with the same set of configuration parameters. Figure 3.16 shows the improvement brought by YuZu compared to ViVo in terms of the overall QoE and its three components (Q_i, I_i^{patch}, and I_i^{frame}, see Eq. 3.7), using all four videos and the users' motion traces.[§] Note that both systems exhibit negligible stall so I_i^{stall} is not plotted. As shown, YuZu brings significant improvement on the average QoE (by 100.6% to 174.9%) and on each QoE component. YuZu outperforms ViVo due to three reasons.

[§]ViVo does not have the notion of patch; instead its basic adaptation unit is a cubic cell. To ensure fair comparisons, we further divide ViVo's cells into virtual "patches" with the same size as YuZu and assign to them its parent cell's corresponding quality level when calculating I_i^{patch}.

First, ViVo does not support SR, which YuZu leverages to boost the QoE. Second, ViVo's viewport adaptation approach becomes less effective when the whole scene appears inside the viewport (which oftentimes appears in our motion traces). SR does not suffer from this limitation. Third, to realize viewport adaptive streaming, ViVo has to perform 6DoF motion prediction, which is error-prone. In contrast, YuZu does not require motion prediction, and therefore exhibits more stable performance in particular when the motion is fast. Note that viewport-adaptation and SR are orthogonal approaches and can be *jointly* applied.

YuZu vs. Simple SR Adaptation. To demonstrate the efficacy of our network/compute resource adaptation design (§3.4.3), we compare it with a simple adaptation approach that differs in two aspects. First, unlike YuZu's *two-stage* adaptation, it only performs *single-stage* adaptation before downloading each chunk. Second, it employs a *deterministic* greedy algorithm that increases the SR ratio of each block within the finite horizon (in chronological order) until the QoE does not further improve. In contrast, YuZu employs a *probabilistic* greedy approach that is less vulnerable to a local maximum. We evaluate the simple adaptation algorithm using our LTE traces (§3.6.4) and plot its result as "Simple" in Figure 3.14. Compared to it, the full YuZu increases the average QoE by 11.4% and reduces the average data usage by 7.9%.

3.6.6 Micro Benchmarks and Resource Usage

We conduct experiments to show the following. (1) The colorization approach of YuZu can indeed produce good visual quality (with a PSNR >38). (2) YuZu can work adaptively with different hardware (we compare the results on 2080Ti and 1660Ti; we also ported YuZu to an embedded system, see Figure 3.10). (3) The main memory (~5GB) and GPU memory (~2GB) usage of YuZu

is acceptable. (4) The (one-time) offline training time is non-trivial but acceptable, and the sizes of SR models are negligible (<0.2% of the video size). The details are presented below. Note that the following micro benchmark results are generated using the PU-GAN model. The results for the MPU model are qualitatively similar.

Quality of Colorization. To evaluate the quality of the colorization step alone, we employ the approach in §3.6.2 where we use PSNR to objectively assess the image quality of rendered viewports. Specifically, we calculate the PSNR values by comparing $\{I_{4\times1}^{NP-Color}\}$ (defined below) with $\{I_{4\times1}\}$ (defined in §3.6.2), using the *Dress* and *Loot* videos and the real users' motion traces (§3.3.2.1). The viewport images of $\{I_{4\times1}^{NP-Color}\}$ are obtained as follows: (1) remove the color from the original (4×1) video; (2) apply the above nearest-point (NP) colorization method to the video generated in Step (1), using the 1×1 video as the low-resolution point cloud stream from which the colors are picked; (3) replay the same motion traces to render the viewport images for the video colored in Step (2). The PSNR values of $\{I_{4\times1}^{NP-Color}\}$ are 38.09±2.44 and 44.15±2.59 for *Dress* and *Loot*, respectively, indicating the high fidelity of colors produced by our method. The above numbers are much higher than the PSNR values in Figure 3.8 (which also includes the colorization step) due to the following reason. PSNR and many other 2D image metrics such as SSIM [230] perform a pixel-wise comparison between two images. In the case of Figure 3.8, a tiny position shift of a 3D point may result in an also tiny position shift of its projected 2D pixel, leading to a pixel mismatch and thus a decreased PSNR score. This problem does not appear in the colorization step.

Impact of Computation-aware Adaptation. 3D SR demands considerable compute resources. Figure 3.17 demonstrates the impact of hardware and computation-aware adaptation, using the *Lab* video. Figure 3.17 considers two GPUs: a more powerful 2080Ti GPU and a less

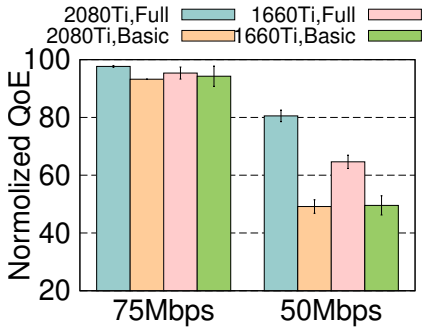


Figure 3.17: Impact of hardware and computation-aware adaptation.

powerful 1060Ti GPU. It also considers two adaptation schemes: the full network/compute adaptation scheme described in §3.4.3 ("Full") and a computation-agnostic scheme that only adapts according to the network bandwidth ("Basic"). The Basic scheme works as follows. (1) It assumes that SR takes no time to complete; (2) it disables 2×2 and 1×4 (otherwise the QoE will degrade too much due to excessive stalls). Under the above setup, each bandwidth setting in Figure 3.17 has four schemes: {2080Ti, 1660Ti} × {Full, Basic}. As shown, when there is sufficient bandwidth, the QoE differences among the four schemes are small, because the player is more likely to fetch 3×1 and 4×1 blocks that do not require SR. However, when the bandwidth becomes low, the difference between 2080Ti and 1060Ti becomes noticeable, and the gap between Full and Basic is even larger. The Basic scheme yields much lower QoE scores because it ignores SR's computation overhead, leading to excessive stalls.

Memory Usage. We measure the client-side memory usage when streaming the *Lab* video over 50Mbps bandwidth (which leads to extensive invocations of SR). On the 2080Ti (1660Ti) desktop, the peak main memory usage is 5.03GB (5.33 GB); the peak GPU memory usage is 1.97 GB (1.83 GB). YuZu's GPU memory usage on 2080Ti is higher than the numbers reported in Figure 3.9 because YuZu loads multiple SR models at runtime. When the available bandwidth is higher, the CPU/GPU memory will reduce because of fewer SR operations.

Offline Training Time and Model Size. YuZu incurs non-trivial model training time. For example, on the 2080Ti desktop, it takes about 88 minutes to train the 1×2, 1×3, and 1×4 models altogether for the *Lab* video consisting of 3,622 frames. However, note that (1) this is a one-time overhead; (2) we did not conduct any performance optimization for training; for a large-scale deployment, the training overhead could potentially be reduced by training one generic model and fine-tuning it for each specific video [247] (left as future work). The SR model size is negligible ($< 0.2\%$) compared to the video size.

3.7 Summary

In this chapter, we conduct an in-depth investigation on applying 3D SR to streaming volumetric content. Our proposed QoE model and the YuZu system take a first and important step toward making SR-enhanced volumetric video streaming principled, practical, and affordable. YuZu demonstrates how a series of novel optimizations, which fill a 500× performance gap, as well as judicious network/compute resource adaptation can help significantly improve the QoE for volumetric video streaming.

Chapter 4

Habitus: Boosting Mobile Immersive Content Delivery through Full-body Pose Tracking and Multipath Networking

4.1 Introduction

Now, let us extend our focus from volumetric videos to broader immersive content such as virtual/mixed reality. As discussed in §2.2, the huge network resource requirement of immersive content poses a major challenge for mobile immersive content delivery systems, which use wireless radios instead of HDMI/USB cables [219, 218] for content delivery. Though recent advancement of millimeter wave (mmWave) radio technologies make it feasible to transmit immersive content at a multi-Gbps data rate, our case study in §2.2.1 shows that simply using mmWave for immersive content delivery may not help improve QoE, and all existing solutions, including *improving the PHY layer*, *enhancing line-of-sight (LoS)*, and *using specialized equipment*, have limitations.

In this chapter, we investigate two complementary, under-explored dimensions to improve the performance of mmWave-based immersive content delivery systems: (1) *full-body-pose guided mmWave throughput prediction* and (2) *joint use of mmWave and omnidirectional radios*. We then integrate them into a holistic middleware framework called Habitus. At a high level, Habitus features a judicious cross-layer design that considers the interplay among viewers' motion, wireless networks, and immersive applications. It creatively leverages features on cheap commodity mobile devices (*e.g.,* dual 802.11ac/ad radios and multi-lens cameras capable of producing stereo images) for affordable high-quality immersive content delivery. Habitus is readily deployable without requiring any change to the existing wireless protocol stack, hardware, or driver. It is orthogonal to and can co-exist with the three categories of solutions described above. The **key challenges** we face include: (1) the dynamics of viewers' motion and mmWave channel incur complex interplay, making accurate throughput prediction difficult; (2) diverse locations and human viewers add more complexity in developing a robust prediction model; even at the same location, the environment may change (*e.g.,* a moved chair or a walking spectator); (3) the heterogeneous characteristics of mmWave and omnidirectional radios make their duet difficult.

Full-body-pose Guided mmWave Throughput Prediction (§4.3). Over a mmWave link, although the throughput fluctuations cannot be completely avoided, they can potentially be predicted to improve the quality-of-experience (QoE) of immersive applications. Habitus utilizes not only the network information, but also viewers' motion to predict mmWave performance. The rationale is that by continuously tracking the 6-DoF motion, an immersive content delivery system can estimate the viewer's future motion trajectory [166, 68, 241], which can then be mapped to the future mmWave performance given the sensitivity of mmWave signal to the physical environment. In particular, we make a new discovery that using the viewer's *full-body*

pose (how a person stands, sits, or moves as represented by a set of key points associated with body parts/joints) as features can significantly improve the throughput prediction accuracy, due to the *spatial correlation among body parts* during typical human motion [17]. Motivated by the above, we develop a first-of-its-kind framework that predicts mmWave throughput by jointly leveraging a headset's 6-DoF motion, the viewer's body pose, and network information, through a unified machine learning model. The full-body pose can be captured by a commodity stereo camera conveniently placed, *e.g.,* next to the WiFi AP.

Reacting to Unseen Changes (§4.4). Using a pre-trained model to predict throughput suffers from a key limitation: it cannot adapt to changes deviating from the training data. We systematically investigate how various types of changes in the immersive streaming context impact the prediction accuracy of a pre-trained model. Based on the insights, we design three orthogonal mechanisms for reacting to different types of unseen changes: (1) *offline transfer learning* handles large changes such as switching to a new location/user; (2) *online transfer learning* updates the model at runtime to tackle smaller changes such as new motion patterns and environmental perturbations; (3) we also leverage the stereo camera to *proactively detect/respond to moving objects* (*e.g.,* a passing person) that affect the mmWave performance.

Joint Use of mmWave and Omnidirectional Radios (§4.5). Multi-band radio access is a common feature on both mobile devices and WiFi APs. For example, the Asus ROG Phone Series [176] support both 802.11ac and ad. Strategically combining them can boost the network performance for metaverse. We design a lightweight yet effective multipath scheduler for immersive content delivery over mmWave and omnidirectional radios (802.11ad and ac in our prototype). It employs two core design ideas. First, it prioritizes the (low-bandwidth but stable) ac path to

guarantee the basic user experience, and opportunistically leverages (high-bandwidth but fluctuating) ad whenever possible. Second, it enhances the mmWave throughput prediction through robust statistical trend analysis [39] to facilitate longer-term throughput forecast.

Implementation (§4.6). Instead of building a monolithic application, we develop the above features as a generic, user-space middleware framework called Habitus. It offers simple interfaces and data-handling paradigms that are compatible with a wide range of existing immersive applications. It also addresses practical system-level challenges, such as accurate throughput measurement of the highly bursty traffic of immersive content delivery. Habitus consists of 3,541 lines of code (LoC). To demonstrate its efficacy, we develop two immersive apps using its API: one is built from scratch in 5.2K LoC; the other is adapted from a state-of-the-art volumetric streaming system [68] by only modifying 47 LoC.

Datasets (§4.3, §4.4) and Evaluation (§4.7). We thoroughly evaluate Habitus through real-world data and deployment.

• We conduct IRB-approved data collection involving 10 representative motion patterns at 4 representative indoor locations from 3 users. This results in a 21-hour dataset that was used to evaluate Habitus's prediction framework.

• We enhance the above dataset with both static and dynamic environmental changes in a reproducible manner (*e.g.,* using a robotic arm to programmatically inject NLoS, see our demo video [43]), to evaluate Habitus's reaction to changes.

• Using full-body poses reduces 802.11ad throughput prediction error by up to 29% (25%) in MAE (RMSE), compared to using only 6-DoF head motions. This translates to an average QoE improvement of 29% for volumetric content delivery.

• Habitus effectively responds to unseen changes. The offline transfer learning reduces the model training time by 36% to 55% compared to building the model from scratch when switching to a new location or user. The online transfer learning can adapt to a new motion pattern or a typical static environmental change in 32 secs and 15 secs, respectively. By proactively detecting and responding to moving objects, Habitus reduces the volumetric streaming stall by 7%.

• Our multipath solution boosts the average volumetric video quality by 67%, reduces the stall by 64%, and improves the QoE by 72%, compared to using 802.11ad alone. Compared to a recent multipath solution for 802.11ac/ad [183], Habitus reduces the stall by 58% and boosts the quality by 19%.

• We conduct another IRB-approved user trial where we collect 12 viewers' subjective feedback when watching volumetric content. The average ratings for 802.11ad only (basic prediction), 802.11ac+ad (no ad prediction), 802.11ad+ac (basic ad performance prediction), and the full Habitus system (with multipath and full-fledged ad prediction) are 2.67, 2.75, 3.08, and 3.50, respectively (in a 1–5 scale).

Habitus represents to our knowledge a first software framework aiming at optimizing the upper-layer network protocol stack for immersive content delivery (and metaverse applications in general). This paper makes three-fold contributions: the design of the Habitus framework; its implementation, evaluation, and integration into two volumetric content delivery systems; and the release of data [67] (802.11ac/ad performance correlated with full-body motion) and source code [66].

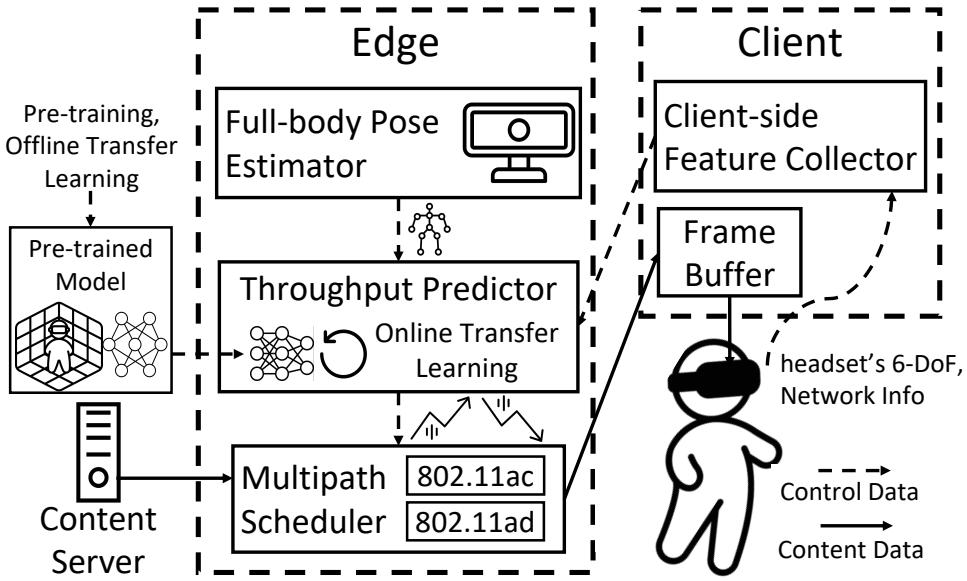


Figure 4.1: The system architecture of Habitus.

4.2 Habitus **Overview**

Habitus is a software framework enabling immersive content delivery applications to better interact with heterogeneous off-the-shelf wireless networks. It offers two essential features: *accurate runtime mmWave throughput prediction* and *multipath networking* over mmWave (*e.g.,* 802.11 ad) and omnidirectional radio (*e.g.,* 802.11 ac). We assume that the bottleneck is the last-mile radio link(s). Figure 4.1 shows the workflow of Habitus. As the viewer is watching immersive content, Habitus collects various features in real-time and sends them to an edge node. The edge employs a machine learning model to perform accurate mmWave throughput predictions. The prediction results are then utilized by the application and Habitus's multipath scheduler to determine the appropriate content quality level and how to distribute the content over diverse radio links, respectively. Habitus works on top of the transport layer as a middleware. It is ready for deployment without requiring any changes to the existing wireless protocol stack, hardware, or drivers.

Developing Habitus brings us several challenges: (1) **How to accurately predict mmWave throughput at runtime?** Due to its sensitivity to blockage and mobility, mmWave throughput is difficult to predict, especially under user mobility. (2) **How to ensure the robustness of the prediction models?** A pre-trained model may not be able to handle various unseen changes such as moving to a new location, switching to a different user, or even smaller environmental perturbations. (3) **How to make multipath scheduling efficient and intelligent?** mmWave and omnidirectional radios have their unique natures. How to strategically use these properties to facilitate multipath scheduling is another critical problem.

Habitus tackles the above challenges with 3 core designs.

• **Full-body-pose Guided mmWave Throughput Prediction (§4.3).** Habitus boosts mmWave throughput prediction by exploiting the viewer's full-body pose, along with the headset's 6-DoF motion and network information, as the features. It employs a cheap stereo camera with off-the-shelf computer vision techniques to capture viewers' poses. We systematically demonstrate the benefit of leveraging body pose through a 21-hour dataset collected at 4 diverse locations.

• **Reacting to Unseen Changes (§4.4).** Habitus reacts to unseen changes in training data via three orthogonal approaches: offline transfer learning for location/user changes, online transfer learning for new motion patterns and small environmental changes, and proactively detecting/responding to moving objects (*e.g.,* a passing person) affecting mmWave performance.

• **Joint Use of mmWave and Omnidirectional Radio (§4.5).** Habitus employs the omnidirectional radio that provides stable throughput as a basis. It then opportunistically takes advantage of the fluctuating mmWave radio. It exposes simple, generic interfaces to a wide range of immersive applications.

4.3 mmWave Throughput Prediction Guided by Full-body-pose

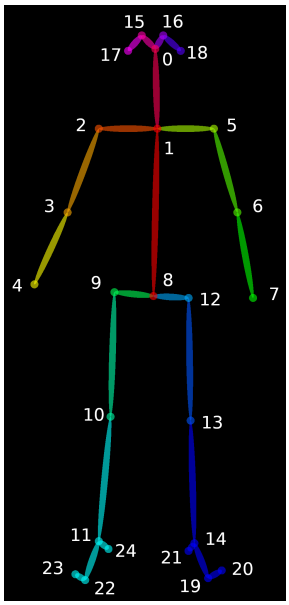
Habitus utilizes not only the network information, but also viewers' motion as features to predict mmWave throughput. It is based on our two observations. (1) The fast signal attenuation and vulnerability to LoS blockages make mmWave throughput highly correlated with the headset's physical position and orientation [237, 147]. Both of them can be predicted from the viewer's historical 6-DoF motion trajectory [68]. (2) Various body parts exhibit spatial correlation [17]. It provides opportunities to enhance the headset motion prediction, which can facilitate mmWave throughput prediction.

4.3.1 Full-body Pose Estimation

4.3.1.1 Full-body Pose Representation and Retrieval

Typically, a full-body pose can be represented by a set of key points where each key point corresponds to a joint/part of the human body. We customize the OpenPose [31] BODY_25 format – a popular format used in the computer vision community – to represent the full-body pose. As shown in Figure 4.2, we discard some key points (*i.e.,* eyes, ears, toes, and heels) that have little contribution to the full-body pose. We keep the other 15 key points to represent the viewer's full-body pose, covering the nose, neck, shoulders, elbows, wrists, hips, knees, and ankles.

Some commercial products (*e.g.,* smart suit [177] and body-mounted sensors [220]) allow tracking the full-body pose, but they are expensive and uncomfortable to wear. Habitus instead employs a cheap and easy-to-deploy approach to capture/track the viewer's full-body pose



Used	Discarded
0: Nose	15: Right Eye
1: Neck	16: Left Eye
2: Right Shoulder	17: Right Ear
3: Right Elbow	18: Left Ear
4: Right Wrist	19: Left Big Toe
5: Left Shoulder	20: Left Small Toe
6: Left Elbow	21: Left Heel
7: Left Wrist	22: Right Big Toe
8: Mid Hip	23: Right Small Toe
9: Right Hip	24: Right Heel
10: Right Knee	
11: Right Ankle	
12: Left Hip	
13: Left Knee	
14: Left Ankle	

Figure 4.2: OpenPose BODY_25 Format.

through a stereo camera. It first applies a machine learning model [31, 11, 21, 170, 129, 28] to the RGB frame to estimate the 2D key points of the full-body pose, each comprised of a 2D coordinate and a confidence value w ($0 \leq w \leq 1$). Habitus then maps the 2D key points to 3D space using the depth map [61] that is generated from stereo images, and keeps their confidence values unchanged.

4.3.1.2 Estimating Missing Key Points

In some cases, for example, when some parts of the body are outside the stereo camera's viewport, we are not able to capture their key points. We observe from our dataset in §4.3.2 that for 86% of time, the ML model can retrieve at least 10 (out of 15) key points. The 90-th percentile, mean, and median duration of a key point's missing time are 1s, 0.43s, and 0.07s, respectively. This indicates that in most cases, a key point misses for a very short duration. We estimate a missing key point's 3D coordinate on the fly using a combination two approaches: reusing its most recently captured

3D coordinate (when the missing period is short), and linearly extrapolating its coordinate using its historical trajectory (when the missing period is long). The detailed design and evaluation are in the following.

To estimate a missing key point's 3D coordinate on the fly, we consider two baseline approaches: simply reusing its most recently captured 3D coordinate (Method 1), and linearly extrapolating its coordinate using its historical trajectory (Method 2 [99, 238, 41]). To assess them, we select a subset of our dataset with no missing key point (as the ground truth, referred to as D_g). From D_g, we create a dataset D_a where key points are removed for k consecutive frames where k is exercised from 1 to 60. We apply the above two approaches to D_a, and find that when the missing duration is short (long), Method 1 (Method 2) gives a lower average estimation error (RMSE). This finding leads to our solution where we switch between the two baselines based on the missing duration. The switching threshold is empirically set to 7 or 14 frames for 30 and 60 FPS respectively, based on the data.

To evaluate our solution, we construct another dataset D_b from D_g. In D_b, key points are removed in such a way that their missing time follows the same distribution as that in our entire dataset. Compared to using the two baselines alone, our solution reduces the average RMSE by 21% and 15%, respectively.

Recall from §4.3.1.1 that a key point contains a confidence value w. We gradually decay w as a key point remains absent, because as the missing time t increases, its 3D coordinate estimation becomes less reliable. We let $w(t) = w_0 \times max(0, 1 - \frac{t}{T})$ where w_0 is the most recently captured confidence value of this key point and T is a threshold controlling the decay speed. We empirically set T to 1 sec, *i.e.,* the 90-th percentile missing time for a key point in our dataset. The confidence value will be used in §4.3.3 as an input to the prediction model.

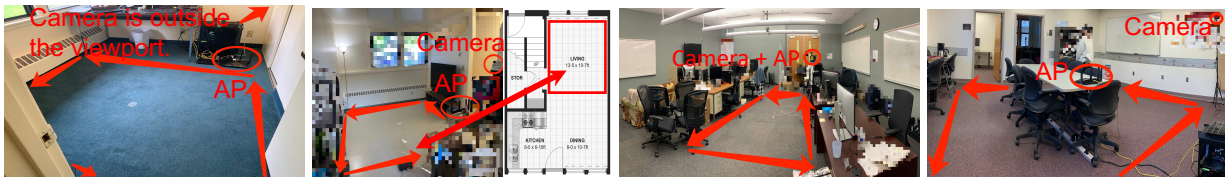


Figure 4.3: Data collection locations (from left to right: *Personal Office*, *Living Room*, *University Office*, and *Meeting Room*).

4.3.2 Data Collection

We perform a first-of-its-kind study on full-body-pose assisted throughput prediction in real-world settings. We conduct an IRB-approved data collection involving 10 motion patterns at 4 indoor locations from 3 users, resulting in a 21-hour dataset consisting of both the network data and viewers' motion data. This unique dataset is used to evaluate mmWave throughput prediction (§4.3.3), techniques for tackling unseen changes (§4.4), and the Habitus system (§4.7).

4 Indoor Locations. We investigate 4 representative indoor locations with diverse environments. As shown in Figure 4.3, the four data collection locations we select (*Personal Office*, *Living Room*, *University Office*, *Meeting Room*) have diverse environments in terms of the layout, floor materials, furniture types, and spatial openness, *etc.* Table 4.1 summarizes the moving area and room space of the four locations. The moving area is the area for data collection. The room space refers to the total space of the entire room. The data collection area of *Personal Office* covers almost the entire room. While *Living Room* also has a similarly simple setup, its data collection area only covers one-third of the room and appears more open, as shown in the floor plan. *University Office* is a large room with a complex layout. *Meeting Room* has a long table in the center of the data collection area. The relative positions between the camera and the WiFi AP also differ across the four locations.

Location	Moving Area (m^2)	Room Space (m^3)
Personal Office	2.5×2.5	3.3×2.5×2.5
Living Room	3.0×2.7	6.6×5.4×2.5
University Office	4.0×2.0	9.0×7.0×3.0
Meeting Room	4.0×3.0	6.5×6.5×2.8

Table 4.1: The moving area and room space of the four locations.

3 Users. We recruit 3 users with different heights (1.6m, 1.7m, and 1.8m) and genders (1 female and 2 males) to collect data in all the four locations.

10 Motion Patterns. We consider 10 representative motion patterns when watching immersive contents [68] and summarize them in Table 4.2. For each motion pattern, we repeat data collection three times.

Dataset Overview. Our dataset consists of 12 {Location, User}-specific sub-datasets, each having 30 (10 motion patterns×3 repeats) data traces. The duration of each data trace is 120 secs and the time granularity of each data point is $1/60$ secs. Our dataset consists of not only the network information (throughput and signal strength of both 802.11ac and 802.11ad), but also users' motion information (*i.e.,* 6-DoF motion of users' headsets and users' full-body pose, see §4.3.1). Across all data traces, the average 802.11ad throughput varies from 275 to 886 Mbps, with the standard deviation ranging from 85 to 358 Mbps. Meanwhile, the average 802.11ac throughput varies from 175 to 378 Mbps, with the standard deviation ranging from 26 to 86 Mbps. The highest throughput of 802.11ad only achieves 2.34× of 802.11ac due to the limitation of our hardware setup.

Hardware Setup. We take *Personal Office* in Figure 4.3 as an example. **For the edge**, we set up a desktop PC with two network interfaces (NICs) at the corner of the room. It has an Intel Core i9-10900X CPU @ 3.70GHz, an NVIDIA 2080Ti GPU, and 32GB memory. Each NIC is connected

Patterns	Description
S1	The user stands in the center of the room, turning around in a clockwise direction.
S2	The user stands in the center of the room, turning around in a counterclockwise direction.
S3	The user walks around in a clockwise direction.
S4	The user walks around in a counterclockwise direction in a normal speed.
S5	The same as S4, but in a slow speed.
S6	The same as S4, but in a fast speed.
S7	A chair occupies the front place of the access point. The user walks around in a counterclockwise direction.
S8	The same as S3, but the user does not change the orientation of his/her head.
S9	The same as S4, but the user does not change the orientation of his/her head.
S10	The user walks around following the walking trace in S7, but there is no chair.

Table 4.2: User motion patterns.

to an access point by a 1-Gbps Ethernet cable, one [212] for 802.11ac and the other [144] for 802.11ad. The two* APs reside on the floor side-by-side. A stereo camera [251] is installed on the wall and connected to the PC via a USB 3.0 cable. It captures users' motion as RGB-D videos at up to 100 FPS. **For the client device**, users wear a headset [126] for collecting 6-DoF motion of their heads. We mount a smartphone [176] that supports both 802.11ac/ad on the headset to collect network information. We use the same hardware setup for all four locations.

Data Collection Methodology. On the client side, the smartphone establishes two TCP connections with the edge over 802.11ac and 802.11ad, respectively. It performs bulk download from the edge through both paths, and measures the throughput and signal strength (in terms of RSSI). The headset keeps sending the 6-DoF to the smartphone by UDP over 802.11ac. Meanwhile, on the edge side, the desktop PC keeps sending the most recent RGB-D frame ID captured by the

*Ideally one access point is able to handle both ac and ad. We use two access points due to the 1-Gbps speed limitation of our Ethernet cables.

stereo camera to the smartphone through UDP over 802.11ac for data synchronization purpose (see next).

Data Synchronization and Post-processing. During data collection, the smartphone logs all the information (generated by itself or received from the headset/edge) except the RGB-D video that is recorded at the edge. Since all the devices are in close proximity, the UDP one-way delay is negligible (≤ 2ms) so different pieces of data are properly synchronized. We use zed-openpose [253] (which consumes the RGB-D video) to estimate the user's body pose offline. The pose is synchronized with other information through the RGB-D frame ID, which is recorded by the client at runtime. For key point extraction, we set the input resolution to 320×240 and keep it consistent in our implementation (§4.6).

4.3.3 Prediction Methodology and Evaluation

We first formulate our prediction task. Let x_t denote the feature vector at time t, y_t denote the predicted throughput at time t, Δt denote the time granularity, and $\mathbb{M}$ denote the prediction model. Assuming that we perform prediction at time t_0, we have

$$Y_{t_0,m} = \mathbb{M}(X_{t_0,n}) \tag{4.1}$$

where

$$X_{t_0,n} = [x_{t_0-(n-1)\times\Delta t}, x_{t_0-(n-2)\times\Delta t}, \ldots, x_{t_0-1\times\Delta t}, x_{t_0}] \tag{4.2}$$

is the feature sequence within a history window (*hw*) n, and

$$Y_{t_0,m} = \begin{cases} [y_{t_0+m\times\Delta t}] & \text{or} \\ [y_{t_0+1\times\Delta t}, y_{t_0+2\times\Delta t}, \cdots, y_{t_0+(m-1)\times\Delta t}, y_{t_0+m\times\Delta t}] \end{cases} \tag{4.3}$$

can be either a single predicted value after a prediction window (*pw*) m or a predicted sequence within the *pw* m where $y_{t_0+i\times\Delta t}(1 \le i \le m)$ corresponds to a future timestamp $t_0 + i \times \Delta t$. The *hw* (*pw*) in secs is computed as $n \times \Delta t$ ($m \times \Delta t$).

Habitus uses the full-body pose by taking the coordinates and confidence values (§4.3.1) of its key points as important features to the mmWave throughput prediction models. We investigate 5 different models (a traditional machine learning model and 4 deep learning models) from recent studies on mmWave throughput prediction [142, 5]. We customize them to our prediction task by tuning the model architecture and the parameters. We list them as follows.

(1) Gradient Boosting Decision Tree (GBDT) [142]. GBDT has been used for predicting commercial 5G throughput [142]. It employs a weighted combination of weak learners that optimizes a differentiable loss function in functional space. Our *GBDT* model has 100 estimators, bounded by a depth of size 3. It takes $X_{t_0,1}$ as the input and predicts a single throughput value $Y_{t_0,m}$.

(2) Fully-connected Neural Network (BP) and Recurrent Neural Network (RNN) [5]. Prior work [5] uses BP and RNN for 802.11ad throughput prediction in a restricted setting (mounting the device on a guided rail with only 2-DoF). In contrast, we apply them to our setting where the device exercises 6-DoF motion. *BP8* is a fully-connected neural network with 3 hidden layers, each with 40 neurons. It takes $X_{t_0,8}$ as the input and predicts a single throughput value $Y_{t_0,1}$.

RNN8 and *RNN20* have the same network architecture, *i.e.,* a recurrent neural network with 3 hidden layers, each with 8 or 20 neurons. They take $X_{t_0,8}$ and $X_{t_0,20}$ as the input, respectively, and both predict a single throughput value $Y_{t_0,1}$.

(3) Sequence-to-sequence Learning (Seq2Seq) [202, 215, 37, 142]. Seq2Seq learning allows inferring an output sequence with variable length from an input sequence. It has also been applied to commercial 5G throughput prediction. Our *Seq2Seq* model has a single-layer LSTM encoder-decoder architecture with 128 hidden units. It takes $X_{t_0,n}$ as the input and predicts future throughput sequence $Y_{t_0,m}$ where $m = 2 \times n$. We get the relationship between n and m from [241], which applies deep learning to predicting viewers' head movement when watching 360° videos. We experimentally confirm that their configuration also works well in our setting.

We use our dataset (§4.3.2) to train/evaluate the above models {*w/*, *w/o*} the full-body pose as extra features. We perform 10-fold cross-validation for each model on each {Location, User}'s dataset, and quantify their prediction errors by mean absolute error (MAE) and root mean square error (RMSE). For *Seq2Seq*, to fairly compare it with the other models, we only use the value y_{t_0+pw} in its predicted sequence. We use three prediction windows (*pw*) of {0.5, 1, 2} secs.

Figure 4.4 shows the average MSE and RMSE for different models across all {Location, User}'s datasets when pw=1 sec. The model trained with (without) full-body pose is denoted as *Model w/ Pose* (*Model*) in Figure 4.4. We also normalize the prediction error by the average 802.11ad throughput (493 Mbps) of our dataset. We have four observations here. (1) *Seq2Seq w/ Pose* achieves the lowest prediction error: 31 (47) Mbps in MAE (RMSE). (2) Leveraging full-body pose as extra features effectively reduces the prediction error for all the models. The reduction ranges

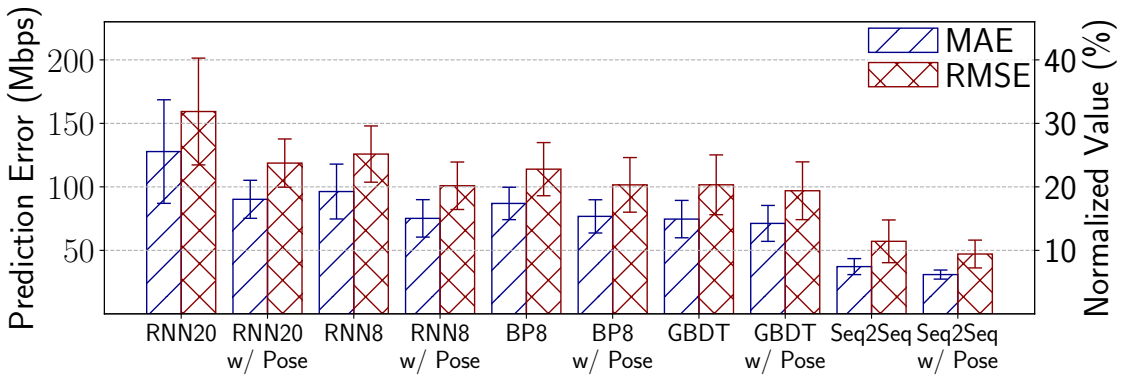


Figure 4.4: Prediction error of different models {*w/, w/o*} full-body pose as extra features, pw = 1 sec.

from 5% (*GBDT*) to 29% (*RNN20*) in MAE and 5% (*GBDT*) to 25% (*RNN20*) in RMSE. This quantitatively confirms that leveraging spatial correlation among body parts helps boost the throughput prediction accuracy under viewers' constant motion. (3) Although not shown, the benefit of leveraging full-body pose is similar in both simple (*i.e., Personal Office* and *Living Room*) and complex (*i.e., University Office* and *Meeting Room*) environments. (4) Deep learning models of prior work [5] (*RNN20*, *RNN8*, *BP8*) do not necessarily outperform the non-deep-learning model (*GBDT*). This is likely because those in [5] are designed for limited motion (2-DoF) as opposed to the complex 6-DoF motion in real-world settings. We confirm that the above findings also hold for *pw*=0.5s and *pw*=2.0s.

4.4 Reacting to Unseen Changes

In this section, we investigate how the throughput prediction models developed in §4.3 react to changes unseen in the training data. This is an important aspect we must consider since the changes are common in practice (*e.g.,* new motion patterns, a chair being moved in the room, another person passing by). We then develop solutions to tackle these changes.

4.4.1 Measurement Methodology

Recall from §4.3.3 that we train a model for a particular user's motion patterns in the environment of a particular location. Therefore, a change to any of the above factors may degrade the model's prediction accuracy. We first define these changes.

C1: New Location. The dimensions, interior design, and furnishings of locations differ vastly, as do the locations of the AP and camera. These variations can significantly impact the behavior of mmWave signal propagation [185, 168, 263, 233].

C2: New User. Users differ in their shape and height, resulting in variations of their body poses used by the model.

C3: New Motion Patterns. Even for the same user at the same location, a new motion pattern may lead to unseen trajectories of the head's motion or the body's pose.

C4: Static Environmental Changes such as furniture being moved and even small objects being manipulated can affect the mmWave signal propagation [263, 233]. We do not consider scenarios where, *e.g.,* there are apparent blockages near the mmWave AP. Users should avoid such scenarios.

C5: Dynamic Environmental Changes are similar to **C4** except that the object(s) that perturb the environment are in motion. A representative scenario is that, people as passerby(s) or spectator(s) can temporarily block the LoS and henceforth cause a throughput drop.

We next describe how to measure the impact of the above changes. For a given change C, we construct three datasets $\mathbb{T}_B$, $\mathbb{T}_A$, and $\mathbb{E}^A$. $\mathbb{T}_B$ and $\mathbb{T}_A$ contain the training data *before* and *after* C, respectively; $\mathbb{E}^A$ contains the testing data collected *after* C for evaluating the impact. We use $\mathbb{T}_B$ and $\mathbb{T}_A$ to train two models $\mathbb{M}_B$ and $\mathbb{M}_A$, respectively, using *Seq2seq w/ Pose* with the same

hyper-parameters. Next, we test $\mathbb{M}_B$ and $\mathbb{M}_A$ using $\mathbb{E}^A$, and calculate the corresponding MAE as MAE_B^A and MAE_A^A respectively. Then the impact of C on the throughput prediction accuracy is calculated as $\text{MAE}_B^A - \text{MAE}_A^A$. The tradeoff here is accuracy vs. training overhead: MAE_A^A gives the best accuracy but requires retraining the model; MAE_B^A reuses the old model at the cost of degraded accuracy.

We now describe how to construct $\mathbb{T}_B$, $\mathbb{T}_A$, and $\mathbb{E}^A$ for **C1** to **C5**. Recall from §4.3.2 that our dataset is divided into (3 users) × (4 locations) = 12 groups (*i.e.,* sub-datasets), and each group contains traces of 10 motion patterns. Also, each group's traces are randomly split into training (70%) and testing (30%). Since a model is created for a given (user u, location l) pair, we measure the impact on a per-group basis, and then average the impact across all groups. For **C1**, for a given group (u, l), $\mathbb{T}_B$ consists of the training data of (u, l); $\mathbb{T}_A$ and $\mathbb{E}^A$ contain the training and testing data of $\cup_{l' \neq l}(u, l')$, respectively. For **C2**, it is similar to **C1** except that $\mathbb{T}_A$ and $\mathbb{E}^A$ belonging to $\cup_{u' \neq u}(u', l)$. For **C3**, the three sets all belong to the same (u, l) pair but they contain different motion patterns: $\mathbb{T}_A$ contains all 10 motion patterns; we remove one motion pattern e from $\mathbb{T}_B$, and only keep e in $\mathbb{E}^A$. We repeat the above measurement 10 times, each time using one of the 10 motion patterns as e. For **C4** and **C5**, the three sets all belong to the same (u, l) pair, but we physically introduce the environmental changes and then recollect data for $\mathbb{T}_A$ and $\mathbb{E}^A$, as elaborated next.

We inject two static environmental changes (**C4**) and study them separately: (1) move four chairs in the room to fixed places (Figure 4.5 Left), and (2) put four large packages on designated spots. Then we ask the same users to exercise the same motion patterns (as those in $\mathbb{T}_B$) as if there were no environmental change (we ensure that the changes do not block any motion pattern). Injecting a dynamic change (**C5**) in a reproducible manner requires a more sophisticated

Figure 4.5: Static (left) and dynamic (right) environmental changes, from AP's view.

setup. We use a robotic arm to move a box covered by aluminum foil back and forth between two designated spots to periodically introduce NLoS (Figure 4.5 Right). The box size and aluminum foil thickness are determined through a separate controlled experiment (see next for details) to mimic real humans in terms of NLoS-incurred throughput degradation. In this way, we programmatically emulate a passerby intermittently causing NLoS (demo video [43]). We study **C4** and **C5** only at *University Office* due to setup complexity.

Details of Injecting Dynamic Change. We provide details on how to use a robotic arm to mimic real humans in terms of NLoS-incurred throughput degradation. The idea is to find a material/object that can incur a similar throughput drop to that caused by a real human and is also lightweight enough for the robotic arm to carry. To achieve the above, we perform the following experiment consisting of three steps. (1) We install the smartphone on a tripod and fix it in LoS to the 802.11ad access point. We measure the mmWave throughput when there is no blockage between the smartphone and AP. (2) We ask a real human (height: 1.75 m) to stand between the smartphone and the AP to introduce NLoS and measure the mmWave throughput, which now drops. (3) We ask our volunteer to walk away, and use the robotic arm to hold an object at the same position where the real human stands. We make sure the object blocks the LoS between the AP and the smartphone. We then measure the mmWave throughput and compare

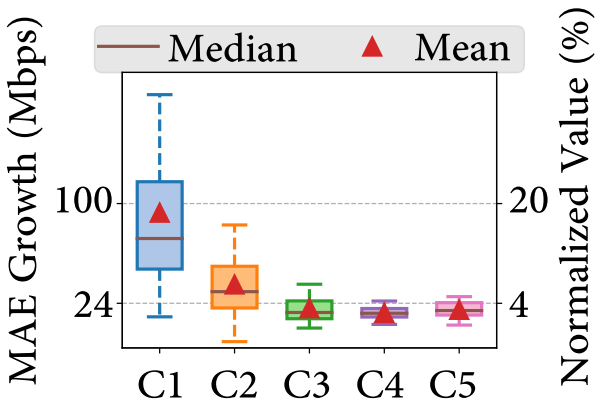


Figure 4.6: Impact of changes on the model prediction accuracy (*Seq2seq w/ Pose*, pw = 1 sec).

it with the measured throughput in Step (2). We want their difference to be small. We try three objects: a paper box, a box covered by an outwear, and a box covered by aluminum foil. We find that a box covered by aluminum foil has the most similar impact on mmWave throughput as a real human (with an average throughput difference of 5%, or 28 Mbps). We therefore use it in our dynamic change experiments.

4.4.2 Measurement Results and Insights

Figure 4.6 plots the impact of **C1** to **C5** on the mmWave throughput prediction accuracy. The two Y axes show both the absolute MAE growth ($\mathrm{MAE}_B^A - \mathrm{MAE}_A^A$) and the relative growth (normalized by the average 802.11ad throughput in our dataset). We highlight our findings next.

A new location incurs the highest impact. C1 degrades the model's accuracy by 19% (93 Mbps) on average. It reveals that the physical property of mmWave and its throughput distribution in a location is the fundamental knowledge learned by our model. Since the indoor location, characterized by its room layout, surface materials, furniture arrangement, *etc.* plays a dominant role in determining the mmWave propagation, changing the location will reshape the throughput distribution at different 3D coordinates.

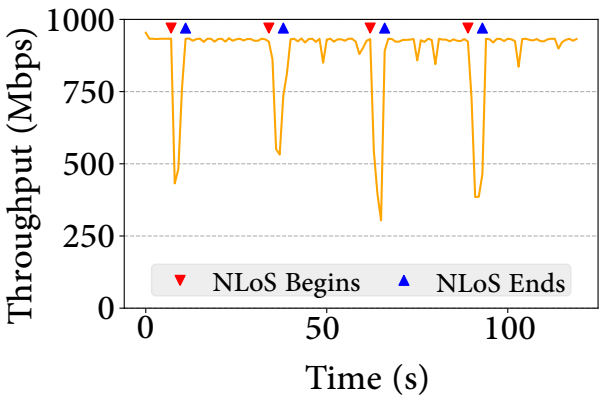


Figure 4.7: mmWave throughput drop caused by dynamic changes (**C5**) in our demo [43].

A new user incurs a moderate impact. **C2** increases the average MAE by 8% (38 Mbps). The impact is lower than that of **C1**, but still non-negligible. This suggests that our model also learns how a user's motion affects the throughput received by the headset. Changing the user alters the relative positions among the key points used by our model; it also changes the trajectory that the headset can reach even with the same motion pattern. Both degrade the model's accuracy. Nevertheless, the impact is lower than **C1** because the throughput distribution, mostly shaped by the surrounding environment, largely remains consistent.

New motion patterns and static/dynamic environmental changes may incur a small impact. Upon **C3**, **C4**, and **C5**, our model's prediction accuracy only drops marginally, by 4%, 3%, 4% (20, 17, 19 Mbps) on average, respectively. The primary reason is that the changes are spatially small (**C3** and **C4**) or temporarily short (**C5**). For **C3**, a new motion pattern oftentimes has positions overlapped with old ones, making the previously learned knowledge relevant. For our **C4** instances, the moved chairs and packages usually do not incur additional NLoS. Our **C5** instance does incur NLoS, but the overall impact is small due to its short duration (Figure 4.7). Note that, however, an environmental change may cause a big mmWave performance impact

(*e.g.,* there are apparent blockages near the mmWave AP). Users of Habitus (and any mmWave system in general) should avoid such scenarios.

4.4.3 Methods of Handling Changes

The results in §4.4.2 suggest that besides taking the time-consuming approach of retraining a model from scratch, Habitus may adopt different strategies to tackle changes. We next introduce three orthogonal mechanisms. The first two (offline/online transfer learning) adopt the concept of *homogeneous transfer learning* [123, 265, 97] that transfers the knowledge learned from a past experience to a new setting. The two properties below make transfer learning a desirable solution. (1) Before and after the change, the feature space (*i.e.,* mmWave signal strength and throughput, 6-DoF motion, full-body pose) remains the same but their distributions (or domains) may differ [265]; (2) before and after a change, there is invariant knowledge (*e.g.,* the physical property of mmWave and the throughput distribution in certain positions) that can be reused. The third mechanism promptly handles **C5** by fusing real-time computer vision into Habitus.

Offline Transfer Learning. For **C1** and **C2**, given their infrequency and non-negligible impact on the model's performance, collecting data under the new setting to update old the model *before using it* helps mitigate large prediction accuracy drops. Specifically, when switching to a new location or a new user, Habitus asks the (new) user to exercise motion patterns (*e.g.,* those in Table 4.2) in the (new) location while measuring the mmWave bandwidth and collecting input features. A typical data collection only needs 1 to 2 minutes (see §4.7.5). Habitus then uses the collected training data to update the old model before starting streaming for the new user or location. Our evaluation (§4.7.6) shows that offline transfer learning reduces the model training

time by 36% to 55% compared to building the model from scratch, which takes about 10 minutes on a medium-end GPU for **C1, C2** and requires much more training data.

Online Transfer Learning. To handle **C3** to **C5** (and also **C1**, **C2**), since they occur much more frequently with usually a much smaller accuracy impact, Habitus can collect data under the new setting and update the model *on-the-fly*. Unlike offline transfer learning, online transfer learning does not incur additional data collection overhead and is transparent to users. Specifically, during a streaming session, Habitus updates the model in consecutive epochs. Epoch i produces a new model M_i and a set of training data samples D_i (input features and the measured ground truth mmWave throughput). Habitus also maintains a global training dataset D_G. At the beginning of Epoch i, Habitus (1) sets the current model for throughput prediction to M_{i-1}; (2) appends D_{i-1} to D_G, and start using D_G to update M_{i-1}; (3) start collecting D_i that will be used to update M_i in Epoch $i+1$. To bootstrap the above process, Epoch 0 only collects D_0 for a fixed period of 10 secs. To avoid D_G becoming too large, Habitus limits D_G to contain only data collected in the recent 5 minutes. We tune the batch size (64) to balance each epoch's convergence speed and the total model copy overhead after epochs. We also apply a small learning rate (0.001) given that we are fine-tuning the model rather than training it from scratch. Our evaluation (§4.7.6) shows that online transfer learning can adapt to a new motion pattern or a typical static environmental change in 32 secs and 15 secs, respectively. Meanwhile, it imposes little impact on the model inference performance since the model is lightweight.

Vision-based Dynamic Change Handling. We find that even online transfer learning is too slow to react to **C5**. We thus devise a heuristic-based design to improve the responsiveness. The idea is to leverage the stereo camera, which already belongs to Habitus's infrastructure, to visually capture dynamic changes and penalize the predicted mmWave throughput accordingly.

Specifically, we focus on the most common dynamic change: a person temporarily blocks the LoS between the viewer and mmWave AP. During a streaming session, the edge performs continuous human detection [26, 222, 252]. If a passerby is detected (we know the viewer's position so the viewer will not be confused with the passerby), Habitus uses the detected 3D bounding box, the known position of the mmWave AP, and 6-DoF motion reported by the headset, to determine if the passerby is causing NLoS or may cause NLoS in the near future by examining the distance from the bounding box center to the LoS between the viewer and AP. If so, Habitus adds an empirical penalty to the mmWave throughput B_{ad} predicted by the model: $B'_{ad} = -\frac{s_{max}-s}{s_{max}-s_{min}} \times B_{ad}$ where s is the observed signal strength; s_{min} and s_{max} denote the typical indoor signal strength range (empirically set to -70 and -30 dBm, respectively). In our evaluation §4.7.6, we show that the above approach reduces the volumetric streaming stall by 7% for **C5**. Meanwhile, the lightweight human recognition model enables a high detection frequency of up to 30 FPS (we use 10 FPS in our prototype).

4.5 System Design of Habitus

We now detail the system design of Habitus that leverages the functionalities introduced in §4.3 and §4.4 as building blocks.

As shown in Figure 4.1, except the client-side feature collector, all the other components of Habitus reside on the edge. This helps minimize the energy consumption and heat dissipation on client devices.[†] The choice of the edge is flexible. It can be either a user's own desktop PC, or an edge node co-located with a mmWave 5G base station (*e.g.,* AWS wavelength [16]). The edge

[†]For example, in our experiment, running our *Seq2Seq* model (§4.3.3) on the ROG phone II [176] for only two minutes will trigger an overheating issue.

also acts as a proxy by forwarding the client's requests to the server and the server's streamed content to the client. Habitus supports both client request and server push. Our prototype uses the former.

4.5.1 Application Interface

Habitus jointly utilizes mmWave and omnidirectional radios (802.11ad and 802.11ac in our prototype) to deliver immersive content. It exposes simple interfaces to applications.

• Through a callback, Habitus keeps informing the application of the two radio links' *aggregated* bandwidth. The application should ensure that its actual streaming bitrate does not overshoot the aggregated bandwidth. The bandwidth update is at a fast pace (*e.g.,* 30 FPS) to match the viewer's fast motion.

• The application streams immersive contents on a per *data block* basis, which can be flexibly defined by the application based on its semantics. Each block can be independently decoded. We use examples in 4.5.4 to show that our block-based paradigm is aligned with the design of many existing immersive apps (*e.g.,* 360° videos, volumetric videos, and VR). When the client requests for (or the server pushes) a block, it uses Habitus API to attach two parameters: the block's *priority* and *playback deadline*. Habitus forwards the blocks based on their playback deadline in a FIFO manner, and distributes high-priority and low-priority blocks over the omnidirectional and mmWave radios, respectively. The rationale is that the omnidirectional radio is more reliable, so high-priority blocks get a higher chance of being delivered (and hence decoded and rendered) before its deadline than low-priority blocks.

4.5.2 Utilizing mmWave Throughput Prediction

Habitus exercises multipath content delivery in two steps. It first estimates the aggregated network bandwidth by treating the multipath ac/ad connections as one logical connection. Second, it splits the block stream over ac and ad paths. We now detail the first step, and describe the second step in §4.5.3.

Recall that the predicted throughput of 802.11ad, denoted as B_{ad}, can be obtained from our mmWave throughput prediction model (§4.3, §4.4). The 802.11ac throughput, B_{ac}, is much more stable and largely not affected by the environment. We therefore simply use the harmonic mean of a past window of 5 secs to predict B_{ac}. Then Habitus predicts the aggregated capacity as $B_{ac} + c \times B_{ad}$. We use c, which we call the *trend-aware coefficient*, to further enhance the 802.11ad throughput prediction by considering the trend of a finite horizon in the future as produced by our model (§4.3.3). The idea is to analyze the monotonic trend of the predicted throughput sequence. If the trend is increasing (decreasing), we can use the 802.11ad link aggressively (conservatively).

To derive c, Habitus first uses Cox-Stuart Test [39, 181], a lightweight, non-parametric approach, to determine the trend (increase, decrease, or neither). The Cox-Stuart Test starts with two statistical hypotheses: (1) H_0: No monotonic trend exists in the series, and (2) H_A: The series is characterized by a monotonic trend, which is further considered as three cases, *i.e.,* (a) an increase or decrease trend exists, (b) an increase trend exists, and (c) a decrease trend exists. Mathematically, in the testing procedure, a throughput sequence $Z = \{z_1, z_2, ..., z_n\}$ (we suppose n is an even number for simplicity) is divided into two parts $\{z_1, z_2, ..., z_{\frac{n}{2}}\}$ and $\{z_{\frac{n}{2}+1}, z_{\frac{n}{2}+2}, ..., z_n\}$. The test statistic $T(+)$ and $T(-)$ is then calculated as $T(+) = \sum_{i=1}^{\frac{n}{2}} \mathbb{I}(z_i < z_{i+\frac{n}{2}})$ and $T(-) = \sum_{i=1}^{\frac{n}{2}} \mathbb{I}(z_i > z_{i+\frac{n}{2}})$, respectively, where $\mathbb{I} \in \{0, 1\}$ is an indicator function. If the null

hypothesis H_0 is true, the statistic $T(+)$ and $T(-)$ should obey the binomial distribution with parameters $\frac{n}{2}$ and $\frac{1}{2}$, *i.e.,* $T(+), T(-) \sim B(\frac{n}{2}, \frac{1}{2})$. Otherwise if $T(+) > T(-)$ (or $T(-) > T(+)$) and the p-value is less than a threshold (*e.g.,* 0.05 in our case), the hypothesis H_A case (b) (or H_A case (c)) is true.

$$c = \begin{cases} 1 + \text{avg}\left(\sum_{i=1}^{\frac{n}{2}} \mathbb{I}\left(z_i < z_{i+\frac{n}{2}}\right) \times \left|\frac{z_{i+\frac{n}{2}} - z_i}{z_i}\right|\right), & \text{if increase trend} \\ 1 - \text{avg}\left(\sum_{i=1}^{\frac{n}{2}} \mathbb{I}\left(z_i > z_{i+\frac{n}{2}}\right) \times \left|\frac{z_{i+\frac{n}{2}} - z_i}{z_i}\right|\right), & \text{if decrease trend} \\ 1, & \text{otherwise} \end{cases} \tag{4.4}$$

Next, Habitus uses the above formula (Equation 4.4) to compute c by averaging the future changes in the predicted throughput sequence $\{z_i\}$. $\mathbb{I}(x) = 1$ iff x is true (otherwise 0), and n is the length of the predicted throughput sequence. The formula splits the sequence into two sub-sequences in the middle, and computes the normalized increase (decrease) of the second sub-sequence compared to the first one, on a per-element basis. The normalized changes are then averaged. We empirically set the lower and upper bound of c to 0.5 and 1.25, respectively.

4.5.3 Multipath Scheduling

Upon receiving the block stream from the server, Habitus splits it over 802.11ac and ad paths. Regarding the splitting mechanism, a straightforward solution is MPTCP [51] or its variants for wireless networks [235, 69, 183]. We reject this design due to three reasons. First, MPTCP exposes to the upper layer a single logical connection whose byte stream is delivered in-order; Habitus instead decouples the two paths that independently deliver blocks. Second, adapting MPTCP's

scheduler to Habitus requires kernel modifications. Third, MPTCP is known to cause issues on ad/ac (*e.g.,* throughput drops over ad due to periodical network scans on ac [183]).

To avoid the above issues, Habitus establishes two single-path connections[‡] and performs scheduling in the user space. The scheduling logic is straightforward: transmitting high-priority and low-priority blocks over 802.11ac and ad, respectively, with the reason explained in §4.5.1. Specifically, the server sends to Habitus's edge node the metadata (headers) of multiple blocks with the same playback deadline in a single bundle, followed by parallel streams of individual blocks' content. As the blocks' content arrives, the edge distributes them over the two paths according to their priority fields in the metadata and each path's estimated bandwidth. A block only usually uses one path, but a small number of blocks may be split over both paths if one path's bandwidth budget is insufficient. Once a block arrives at the client, it will be immediately passed to the application for decoding and rendering. The server-side transmission, edge-side forwarding, and client-side reception are pipelined.

4.5.4 Example Use Cases of Immersive Apps

Habitus can be easily integrated with a wide range of immersive applications and content formats as exemplified below.

360° Videos. State-of-the-art 360° video systems spatially segment each panoramic video chunk into tiles [62, 225, 260]. Tiles are selectively transmitted based on the viewport. Each tile naturally maps to a block in Habitus, and its priority can be set to the probability that it will appear in the viewport. Many existing systems already have this metric calculated [166, 34].

[‡]Our prototype uses TCP. A better design would be using QUIC [101] to avoid head-of-line blocking across blocks within a path under packet losses.

Volumetric Videos. The above viewport adaptation technique and henceforth the assignment of block/priority also applies to volumetric videos, where a 2D tile becomes a 3D cube consisting of 3D points. Alternatively, since each volumetric frame consists of unstructured points, it can be arbitrarily split into multiple layers each constituting a block in Habitus. A "base layer" with a low-density point cloud can be assigned a high priority; one or more "enhancement layers" each encompassing additional details can be assigned lower priorities.[§]

Generic VR. Networked VR systems either stream raw 3D models [254, 118, 102, 258] or rendered 2D scenes [119, 35]. Depending on the content format, a block can be either a 3D model (or part of it) or a rendered 2D patch. The are several studies/systems on determining the priority of VR content, such as those based on foreground/background [118, 100, 239], the viewing distance [57, 139], and user gaze behaviors [35].

4.6 Implementation

Our implementation consists of three parts: (1) the main Habitus middleware in 3.5K LoC; (2) a 802.11 throughput measurement module plugged into Habitus; (3) two sample applications using the Habitus API (5.2K LoC and 4.4K LoC, respectively).

The Main Habitus System is implemented in C++ and Python. We use ROG Phone II [176] and plug it into a low-end VR headset [89] (costs $26) as the client-side device and the same server used in §4.3.2 as the edge node. On the client side, we use Linux `iw` [116] to monitor 802.11 signal strength; we use ARCore [58] for 6-DoF motion tracking (based on IMU and camera data [59]). On the edge side, we use PyTorch-1.10.0 [163] for training and transferring our models. For

[§]A similar concept called Scalable Video Encoding (SVC [186]) can be applied to 2D content, albeit at a higher encoding overhead.

inference, we save the models in TorchScript [210] for C++ execution. We implement the body pose estimator over zed-openpose [253], using a pre-trained model [31, 151] to detect 2D poses. We pipeline the body pose estimation stages (capture, 2D detection, 2D-to-3D mapping). We use the object detection module in ZED SDK 3.8.2 [252] to detect passersby.

802.11 Throughput Measurement Module. Compared to traditional 2D video traffic, immersive content traffic is highly bursty [68, 102, 35]. This poses several challenges for 802.11 (in particular, mmWave) throughput measurement. We thus implement an 802.11 throughput measurement module using Libpcap-1.10.1 [111]. We detail its design in §4.6.1.

Two Volumetric Streaming Applications using Habitus**.** To demonstrate how Habitus can benefit real immersive applications, we build two volumetric (point cloud) streaming systems with different logic and complexity using the Habitus API. The first app (**App1**) employs layered encoding of point clouds (§4.5.4) so each data block corresponds to a (frame, layer) pair. The second app (**App2**) performs viewport adaptation (§4.5.4) by spatially segmenting each frame (*i.e.,* point cloud) into cubical cells, so each data block constitutes a (frame, cell) pair. We build the first app from scratch in 5.2K LoC, and the second app replicating ViVo [68], a state-of-the-art, visibility-aware volumetric streaming system. For ViVo, we only change 47 LoC for Habitus integration. Both apps are equipped with the same bitrate adaptation algorithm whose details can be found in §4.6.2.

4.6.1 802.11ad Throughput Measurement

Compared to traditional 2D video traffic, immersive content traffic is highly bursty. Take volumetric content as an example. First, different from the traditional 2D videos that are encoded

at a group of pictures (GOP) level, volumetric videos are typically encoded on a per-frame basis due to the difficulty of inter-frame encoding. Second, volumetric content players often apply visibility-aware techniques [68, 166, 102, 254, 35] per frame to only download the content inside the viewer's predicted viewport. To maintain accurate viewport prediction results, the client player has to maintain a shallow buffer (*e.g.,* 5 frames in ViVo [68]). Both factors above lead to an extremely frequent request/reply pattern, which renders traditional throughput measurement methods used by 2D video players (simply calculating the ratio between the video chunk size and the chunk download time) very inaccurate. Over mmWave that offers Gbps throughput, the inaccuracy is further deteriorated.

To address the above challenge, we adopt a cross-layer design to measure the throughput by passively examining incoming packets containing immersive content on the client side. Our approach works for both single-path and multipath cases. Specifically, at the application layer, the edge explicitly informs the client how much data will be transmitted over each path before sending data blocks belonging to each frame back-to-back. At the transport layer, the client tracks the arrival time and TCP sequence numbers of the incoming packets. The TCP sequence numbers indicate how much data has been received. Utilizing these information, the client-side throughput measurement module is able to group the *back-to-back* packets in each "burst" as a packet train [143, 171, 237, 250] and use their sizes and timing for throughput measurement. Our approach disregards the ordering and duplicate of packets, and is therefore robust to packet out-of-order and retransmission.

4.6.2 Development of Two Sample Volumetric Streaming Applications

To demonstrate how Habitus can benefit real immersive applications, we implement two sample volumetric content delivery applications with different logic and complexity.

App 1: Simple Volumetric Streaming. We build a simple volumetric streaming system using the Habitus API from scratch in 5,208 LoC. It delivers the volumetric content stored on a Linux server to an Android client over the Internet. The client player uses a shallow buffer of 5 frames (consistent with App 2) for streaming. The content format uses the layered encoding scheme described in §4.5.4: each volumetric frame (point cloud) is split into 64 layers each consisting of non-overlapped points through uniform sampling. Each (frame, layer) pair corresponds to a data block in Habitus's term. The priority of each block is inverse proportional to the number of points in the block. The intuition is to prioritize streaming blocks with sparse points so that the viewer can see the partial content as early as possible.

App 2: Visibility-aware Volumetric Streaming. We also replicate ViVo [68], a state-of-the-art networked volumetric video streaming system. ViVo performs visibility-aware streaming where it only fetches content falling into the viewer's predicted viewport. In ViVo, each volumetric frame (point cloud) is spatially segmented into 64 cubical cells. Each (frame, cell) thus constitutes to a data block in Habitus. The priority of a block is calculated at runtime, *i.e.,* inverse proportional to the Euclidean distance from the center of its cubical cell to the center of the predicted viewport. To integrate Habitus into ViVo, we only change 47 LoC that is mainly for library initialization and blocks transmission/reception.

For both applications, each data block is encoded into 10 quality levels with different point density levels. Both applications use the same throughput-based adaptive bitrate (ABR) algorithm [86] to determine the quality level of each data block. The bitrate selection logic works as follows. Initially, all the to-be-fetched blocks are set to the highest quality level. The ABR algorithm then greedily picks the block with the lowest priority and reduces its quality level by 1. The above process is repeated until the total calculated bandwidth usage does not exceed the aggregated network capacity reported by Habitus, or all the blocks reach the lowest quality level.

4.7 Evaluation

4.7.1 Experimental Setup

Dataset, Devices, and Models. We use the dataset collected in §4.3.2 for our controlled experiments. The devices are the same as those used in §4.6 and §4.3.2. In §4.7.2, we use {*GBDT, BP8, RNN8, RNN20, Seq2Seq*} {*w/, w/o*} *Pose* models, and three prediction windows (*pw*): {0.5, 1, 2} secs. Experiments in other sections use *Seq2Seq* {*w/, w/o*} *Pose* with *pw*=1 sec.

Controlled Experiments. To ensure the reproducibility, during our controlled experiments, we replay the headset's 6-DoF motion traces and the signal strength traces on the smartphone, which is connected to the edge via real 802.11ac/ad links. On the edge side, we replay the RGB-D videos for online full-body pose estimation. We emulate ac/ad throughput traces by Linux `tc` [115]. We fix the smartphone static in LoS to the 802.11ad AP to keep a good mmWave signal for throughput emulation. We do not add additional RTT since the client already connects to the edge via real wireless links.

Volumetric Videos. We use three point-cloud-based volumetric videos (denoted as *V1*, *V2*, and *V3*, respectively) throughout our evaluation. {*V1*, *V2*, *V3*} has {2612, 2700, 3000} frames ({~87, 90, 100} secs), respectively. Each frame of them is split into 64 data blocks (§4.5.1) and details can be found in §4.6 and §4.6.2. All the videos are at 30 FPS, encoded into ten quality levels. The highest bitrates are {570, 687, 738} Mbps for {*V1*, *V2*, *V3*}, respectively. Unless otherwise mentioned, the results reported in the remainder of this section are generated using all three videos.

Roadmap and Metrics. §4.7.2 evaluates 802.11ad throughput prediction error reduction brought by full-body pose. §4.7.3 focuses on the QoE improvement brought by full-body pose. We assess the QoE using the QoE model for point cloud from [254]. It is a linear combination of frame quality, inter-frame & intra-frame quality switch, and stall. §4.7.4 and §4.7.7 evaluate the end-to-end performance (quality and stall) of Habitus. §4.7.5 conducts a user study to examine real users' QoE. §4.7.6 evaluates how our design handles unseen changes. §4.7.8 provides additional micro benchmarks. Except for §4.7.7, we use **App1** in §4.6 (details in §4.6.2) for evaluation.

4.7.2 802.11ad Throughput Prediction Error

Recall that in §4.3.3, we perform 10-fold cross validation for {*GBDT, BP8, RNN8, RNN20, Seq2Seq*} {*w/, w/o*} *Pose* models on each {Location, User}'s dataset, with three *pw*s {0.5, 1, 2} secs. As Figure 4.4 shows, leveraging full-body pose effectively reduces mmWave throughput prediction error for all these models, ranging from 5% (*GBDT*) to 29% (*RNN20*) in MAE and 5% (*GBDT*) to 25% (*RNN20*) in RMSE, respectively.

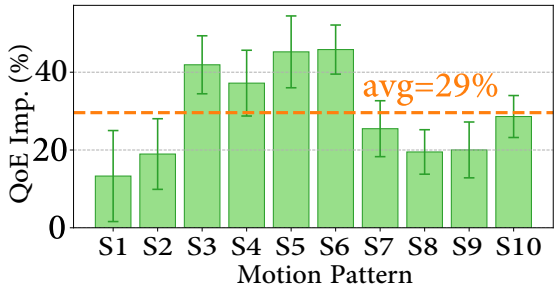


Figure 4.8: Per motion pattern QoE improvement of *Seq2Seq w/ Pose* over *Seq2Seq w/o Pose*.

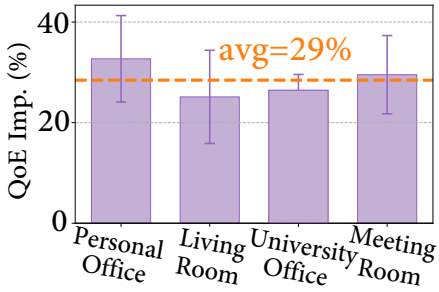


Figure 4.9: Per location QoE improvement of *Seq2Seq w/ Pose* over *Seq2Seq w/o Pose*.

4.7.3 QoE over 802.11ad Network

We evaluate how full-body pose guided mmWave throughput prediction improves **App1**'s QoE through controlled experiments. First, for each {Location, User}, we train a *Seq2Seq w/ Pose* and a *Seq2Seq w/o Pose* model, respectively. We then run **App1** over a single-path 802.11ad network. For each data trace, we run the experiment twice, using *Seq2Seq* {*w/*, *w/o*} *Pose* model, respectively. We log the quality for each data block (§4.7.1) and the stall for each frame to assess the QoE. Figure 4.8 shows the QoE improvement by leveraging full-body pose for each motion pattern across all data traces. We have two findings. First, leveraging full-body pose effectively improves the QoE by 29% on average for all our motion patterns. Second, the QoE improvement varies across different motion patterns, from 13.30% (S1) to 45.82% (S6). The full-body pose does not help much for S1, S2, S8, and S9. This is due to two reasons. First, in S1 and S2, the user does not make translational movement; this reduces the effectiveness of the pose. Second, in S8 and S9, the LoS between the smartphone and the 802.11ad AP is well maintained; this makes the throughput prediction easier compared to other motion patterns. Figure 4.9 presents the QoE improvement for each location in Figure 4.3. The QoE improvement remains similar between simple (*Personal Office* and *Living Room*) and complex locations (*University Office* and *Meeting Room*).

Variant	Predict ad?	*trend-aware*?	Scheduler
ac	No	N/A	single-path ac
ad	No	N/A	single-path ad
Simple	No	N/A	ours in §4.5.3
Pro	*Seq2Seq w/o Pose*	Yes	ours in §4.5.3
Full	*Seq2Seq w/ Pose*	Yes	ours in §4.5.3

Table 4.3: Habitus variants.

4.7.4 End-to-end Performance of Habitus

We evaluate the end-to-end performance, including the content quality and stall, of diverse Habitus variants using all {Location, User}'s data and **App1** in §4.6.

Habitus Variants. Table 4.3 summarizes 5 Habitus variants. We consider two *single-path* variants, *ac* and *ad*, that only schedule data to ac and ad, respectively, without ad throughput prediction. We also consider three *multipath* variants, all using the multipath scheduler from §4.5.3: the *Simple* variant does not utilize 802.11ad throughput prediction; the *Pro* and *Full* variant apply the *Seq2Seq w/o Pose* and *Seq2Seq w/ Pose* model, respectively, for ad throughput prediction. Both *Pro* and *Full* enable the *trend-aware* feature (§4.5.2). As shown in Figure 4.10, compared to *ac* and *ad*, *Simple* boosts the quality (normalized by the highest quality level) by 127.88% and 40.36%, respectively. *Simple* incurs a much higher stall compared to *ac* because the ad network is highly fluctuating and *Simple* blindly uses it without predicting its future condition. Compared to *Simple*, *Pro* boosts the quality by 7.75% and reduces the stall by 44.25%, thanks to the ad throughput prediction and the *trend-aware* multipath scheduler. Compared to *Pro*, *Full* enhances the ad throughput prediction accuracy by using full-body poses, leading to a further stall reduction of 20.55% and video quality improvement of 10.58%.

Habitus vs. Existing Approaches. We compare Habitus with MuSher [183], a recently proposed MPTCP scheduler for ac/ad networks. Musher periodically probes the ratio between the

current ad and ac throughput, and splits the traffic accordingly. In each probe, it tries to increase and decrease the radio, and greedily selects the direction to move based on the aggregated ac/ad throughput measurement. It also has a SCAN component to mitigate the negative impact of network scanning and a BLOCKAGE component to accelerate TCP congestion window recovery after an ad blockage event.

We implement MuSher's scheduling algorithm in the application layer. We plug it into Habitus and denote it as *MuSher-VR*. We do not implement the SCAN component because we use establish separate TCP connections over ac/ad links so network scans on one interface do not affect the other one. We repeat the same experiment on *MuSher-VR*. As shown in Figure 4.10, compared to *MuSher-VR*, Habitus (*Full*) significantly reduces the stall by 58.24% and boosts the quality by 18.52%. Habitus outperforms *MuSher-VR* due to two reasons. First, *MuSher-VR* incurs stalls when it aggressively probes the scheduling ratio by scheduling more data to one path than its actual capacity. Habitus instead takes a prediction-based approach to avoid the stall caused by aggressive probing. Second, Habitus prioritizes using the ac path and opportunistically uses the ad path if possible. In contrast, *MuSher-VR* lacks such prioritization. It schedules the data to the ac/ad paths based on a calculated ratio that ideally should converge to the ratio between ac/ad throughput. However, under the constant movement of the viewer, the actual instantaneous ratio may significantly deviate from the calculated ratio, leading to stalls or under-utilizing the ad path.

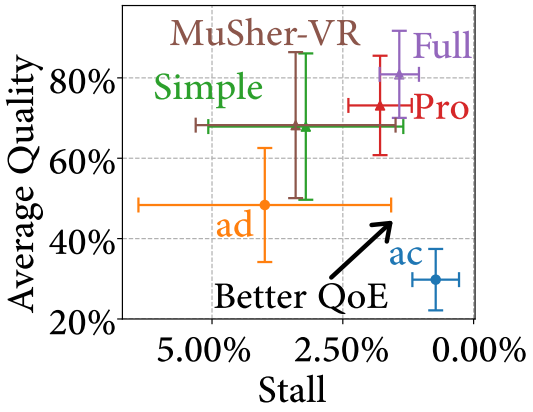


Figure 4.10: Quality vs. Stall of Habitus variants.

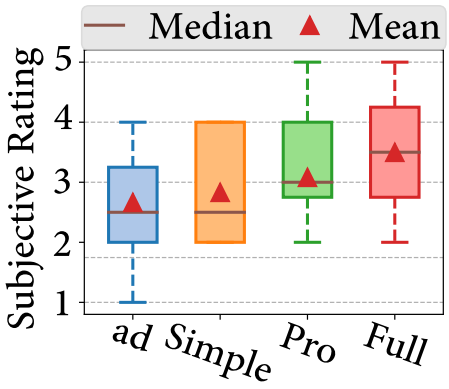


Figure 4.11: Subjective ratings of 12 users.

4.7.5 User Study

We conduct an IRB-approved user study at *University Office* (Figure 4.3) to assess real users' QoE when using **App1** (§4.6). We recruit 12 users with various demographics.[¶] We let each user watch a video randomly selected from our test videos and then subjectively rate the watching experience through 5 choices {1=very bad, 2=bad, 3=fair, 4=good, 5=very good}. Each user performs the above assessment four times. Each time, we randomly plug a Habitus variant into **App1**. We consider four variants: *ad*, *Simple*, *Pro*, and *Full* as listed in Table 4.3. Before each user's trial begins, we collect 2 minutes' worth of data from the user to transfer a pre-trained model to the user. We let the user wear a low-end VR headset [89] with a ROG Phone II plugged into it. The user can freely make 6-DoF motions in the room during the study. As shown in Figure 4.11, compared to {*ad*, *Simple*, *Pro*}, *Full* improves the average subjective rating by {0.83, 0.67, 0.42} (in the scale of 1 to 5), respectively. Note that the best scheme (*Full*) has an average rating of 3.50 (between fair and good), likely because of the hardware limitation of the VR headset (costs $26) compared to a full-fledged VR headset.

[¶]*Gender*: Male: 7, Female: 5; *Height*: <1.65m: 3, 1.65-1.75m: 5, >1.75m: 4. The subjects' ages vary between 20 and 30. 8 out of them do not have prior experience on watching volumetric videos.

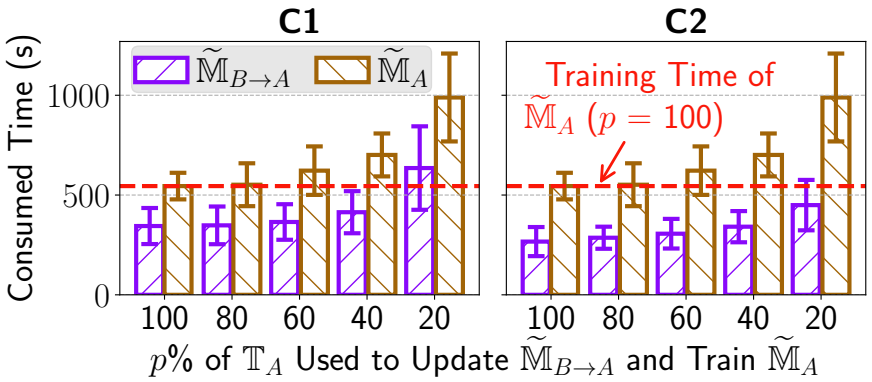


Figure 4.12: Time consumption for (1) transferring from $\mathbb{M}_B$ to $\widetilde{\mathbb{M}}_{B\to A}$ and (2) training a new model $\widetilde{\mathbb{M}}_A$ from scratch after the change, using p% of $\mathbb{T}_A$. (Left: **C1**; Right: **C2**).

4.7.6 Handling Unseen Changes

We evaluate the three techniques introduced in §4.4.3 for handling unseen changes. We reuse the datasets $\{\mathbb{T}_B, \mathbb{T}_A, \mathbb{E}^A\}$ and models $\{\mathbb{M}_B, \mathbb{M}_A\}$ introduced in §4.4.1. The experiments use the *Seq2Seq w/ Pose* model on an NVIDIA 1660Ti GPU.

Offline Transfer Learning. For **C1** and **C2**, we compare the training time between (1) transferring from $\mathbb{M}_B$ to $\widetilde{\mathbb{M}}_{B\to A}$ and (2) training a new model $\widetilde{\mathbb{M}}_A$ from scratch after the change. $\widetilde{\mathbb{M}}_{B\to A}$ and $\widetilde{\mathbb{M}}_A$ denote the transferred model and the built-from-scratch model, respectively. For both $\widetilde{\mathbb{M}}_{B\to A}$ and $\widetilde{\mathbb{M}}_A$, we use $p\% \in \{100, 80, 60, 40, 20\}\%$ of the samples in $\mathbb{T}_A$ to transfer (train) them. Their training stops when the prediction accuracy evaluated on $\mathbb{E}^A$ reaches MAE_A^A (*i.e.,* $\mathbb{M}_A$'s prediction accuracy on $\mathbb{E}^A$). We find that the training always converges even when p is as low as 20%. We show the measured training time in Figure 4.12, where the dashed red line marks the training time of $\widetilde{\mathbb{M}}_A$ with $p = 100\%$. As shown, to achieve the same evaluation accuracy, $\widetilde{\mathbb{M}}_{B\to A}$ significantly reduces the training time by 36% to 41% (48% to 55%) for **C1** (**C2**) across all five p values, compared to $\widetilde{\mathbb{M}}_A$. In particular, training $\widetilde{\mathbb{M}}_{B\to A}$ using only 40% (20%) of the samples in $\mathbb{T}_A$ is still faster than training $\widetilde{\mathbb{M}}_A$ using all the samples in $\mathbb{T}_A$ for **C1** (**C2**). The reason, as

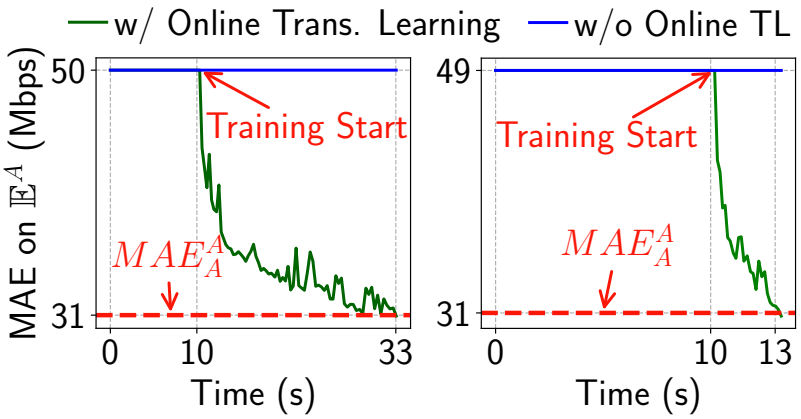


Figure 4.13: Online training cases (Left: **C3**; Right: **C4**).

explained in §4.4.3, is that $\widetilde{\mathbb{M}}_{B\to A}$ effectively reuses the invariant knowledge (*e.g.,* the physical property of mmWave and the throughput distribution in certain positions) that is already present in $\mathbb{M}_B$.

Online Transfer Learning. For **C3** and **C4**, we measure the time consumption when $\widetilde{\mathbb{M}}_{B\to A}$ first converges to the target prediction accuracy MAE_A^A on $\mathbb{E}^A$. To accurately emulate the online setting in a reproducible manner, when training $\widetilde{\mathbb{M}}_{B\to A}$, we feed $\mathbb{T}_A$'s data at the same pace as the real-world training data collection rate. The results indicate that it takes on average 32 (15) secs for the training (*i.e.,* online transfer learning) to converge on **C3** (**C4**), with a standard deviation of 11 (12) secs. The convergence time includes the initial 10-sec bootstrapping (§4.4.3). Figure 4.13 shows case studies for **C3** and **C4**. Note that without online transfer learning, the prediction error on $\mathbb{E}^A$ will never decrease.

Dynamic Change Handling. For **C5**, we evaluate the end-to-end performance of our volumetric streaming app (**App1**) supported by Habitus, *with* and *without* vision-based dynamic change handling (§4.4.3). The controlled experiment is conducted over a single-path 802.11ad network at *University Office* with the robotic arm. We pre-train the vision-based object detection model in a separate experiment so the model can reliably detect the aluminum-foil-covered box

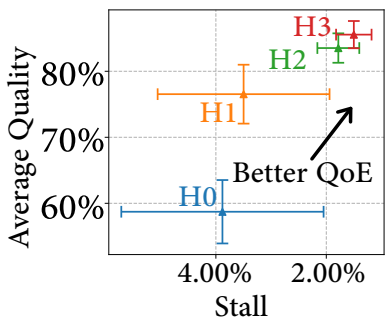


Figure 4.14: Quality vs. Stall of *H0* to *H3*.

H0	Vanilla ViVo [68] w/ single-path 802.11ad
H1	*H0* and multipath 802.11ac and 802.11ad
H2	*H1* and ad throughput prediction *w/o Pose*
H3	*H2* and ad throughput prediction *w/ Pose*

Table 4.4: Apply our solution to ViVo [68] (cumulative).

manipulated by the robotic arm (§4.4.1). We use $\mathbb{M}_B$ as the throughput prediction model. The results indicate that vision-based dynamic change handling reduces the stall by 7% with a video quality reduction of only 2.2%.

4.7.7 Applying Habitus to Existing Systems

We integrate Habitus into **App2** (ViVo [68], an existing volumetric streaming system) by changing only 47 LoC (details in §4.6.2). ViVo adopts visibility-aware streaming by only fetching content that will appear in the future viewport. As listed in Table 4.4, *H0* is our comparison baseline: the vanilla ViVo over single-path ad. *H1* to *H3* involve key components of Habitus. Figure 4.14 shows the quality and stall of *H0* to *H3* across our dataset. As shown, by cumulatively enabling Habitus's components from *H1* to *H3*, both the average video quality and stall improve accordingly. Compared to *H0*, *H3* reduces the stall by 61% and improves the average quality by 46%. In addition, compared to not using pose (*H2*), full-fledged Habitus (*H3*) reduces the stall by 15.75% while slightly boosting the quality by 2.44%. The absolute stall rate of *H3* is 1.67%, meaning that the user encounters less than 0.9 secs of stall per minute on average.

4.7.8 Micro Benchmarks and Resource Usage

We run experiments to show: (1) The GPU memory usage (~4.7G out of 11G on 2080Ti) of Habitus is acceptable. The average processing time of pose estimation and throughput prediction is 27ms and 3.5ms on 2080Ti, respectively. (2) Compared to single-path ac, the additional energy usage and heat increase of **App1** using Habitus are only 1% and $1.3°C$ respectively. The details are in the following.

Resource Usage and Processing Time. For a Habitus-enhanced volumetric content delivery system, the average CPU utilization is 36% on the client side (*i.e.,* ROG Phone II) and 169% (*i.e.,* equivalent to 1.69 cores being fully utilized) on the edge side. The peak GPU memory usage on the edge side is 4721MiB (out of 11GB on 2080Ti) in total, including 2101MiB for video capturing and pose estimation, 1017MiB for 802.11ad throughput prediction, and 1603MiB for object detection. The average processing time on an NVIDIA 2080Ti GPU is 27ms and 3.5ms for pose estimation and throughput prediction using a *Seq2Seq with Pose* model, respectively. The processing time meets the system's requirements.

Energy and Heat. To profile the energy consumption and heat increase of the client device, we run our control experiment using *V2* and {*Personal Office*, User 1}'s data traces repeatedly on a ROG Phone II for 30 minutes. We use **App1** (§4.6) and three Habitus variants {*ac*, *Full*, *MuSher-VR*} in §4.7.4. We start each experiment on a fully-charged phone. After 30-minute running, the battery level drops from 100% to 93% for *ac*, 92% for *Full*, and 92% for *MuSher-VR*, while the device temperature rises from $30.0°C$ to $36.2°C$ for *ac*, from $30.5°C$ to $38.0°C$ for *Full*, and $30.2°C$ to $38.0°C$ for *MuSher-VR*. Compared to *ac*, the additional energy consumption and heat increase of *Full* is 1% and $1.3°C$, respectively. Overall, we believe the resource usage of Habitus is acceptable.

4.8 Summary

Limitations. First, our prototype and experiments only use 802.11ac+ad. We expect Habitus's high-level design to also work with other radio technologies such as 4G + mmWave 5G, but field tests are needed to verify this claim. Second, Habitus's reaction to unseen changes could be further improved. We plan to employ more advanced techniques such as parameter sharing [265] to speed up transfer learning. Third, we focus on the single-user use case. Extending Habitus to multiple viewers will involve additional challenges such as dealing with the interplay among the viewers.

Despite the limitations, in this chapter, we have demonstrated through a working system and rich real-world data that, full-body-pose guided throughput prediction and joint use of omnidirectional and mmWave radios can significantly improve the QoE (up to 72%) for immersive applications. Furthermore, by fusing transfer learning and vision-based object recognition, Habitus can smoothly adapt to unseen changes.

Chapter 5

Alice: Low-latency Image Live Co-editing via Adaptation

5.1 Introduction

In this chapter, we study another emerging networked multimedia applications, *image live co-editing*.

In this paper, inspired by *adaptive video streaming*, we achieve low-latency image live co-editing by incorporating extensive adaptation capabilities into the image LCE system. These capabilities are enabled by various image edit compression and transmission strategies. The key challenge we face, as mentioned above, is that the commonly used *data-based* transmission – where the (compressed) pixel data of an edit is sent to other users – has limited adaptability due to the mathematical constraints of lossless compression [189]. We address this challenge through the following novel **designs**.

• Joint Use of *Data-based* and *Operation-based* Strategies (§5.3, §5.5.1). Our pilot study on real user image editing patterns (§5.3) reveals opportunities to leverage an alternative strategy, referred to as *operation-based*, for image edit transmission, where the editing operation's metadata (*e.g.,* API

and parameters) is sent to other users, who then "replay" the edit on their local copies. To optimize image edit transmission, we strategically switch between *data-based* and *operation-based* approaches based on available bandwidth and computational resources.

• Leveraging Multiple Lossless Compression Techniques with Different Configurations (§5.4, §5.5.2). Numerous lossless compression techniques have been developed, each exhibiting significant variability in both compression ratio and efficiency within the context of image LCE (§5.4). To enhance the adaptability of *data-based* transmission, we strategically combine these techniques and their configurations.

• Real-time Strategy Selection (§5.5.2). We develop a unified lookup table (LUT)-based approach for rapid selection of the optimal transmission strategy, built through extensive offline profiling.

We **implement** the above components into Alice, a latency-aware, cross-platform image edit transmission framework. To evaluate Alice, we integrate it into our cloud-based image LCE testbed. Our extensive trace-driven evaluation (§5.7) demonstrates that: (1) Compared to baselines with fixed transmission strategies, Alice achieves up to 95% per-tile latency reduction, with an average (median) per-tile latency of 160ms (85ms); (2) Across diverse system setups (e.g., varying tile resolutions and user counts), Alice consistently outperforms baseline strategies, delivering 28%–85% latency reduction; and (3) Alice incurs negligible overhead (< 1 ms) on commodity devices (Ubuntu desktop and MacBook Air 2020).

Our **contributions** include: (1) To the best of our knowledge, this is the first study to focus on low-latency image live co-editing, addressing the problem from a system perspective; (2) The design of Alice, including its hybrid transmission strategy and the LUT-based adaptation algorithm; and (3) The implementation, integration, and thorough evaluation of Alice on our self-developed testbed.

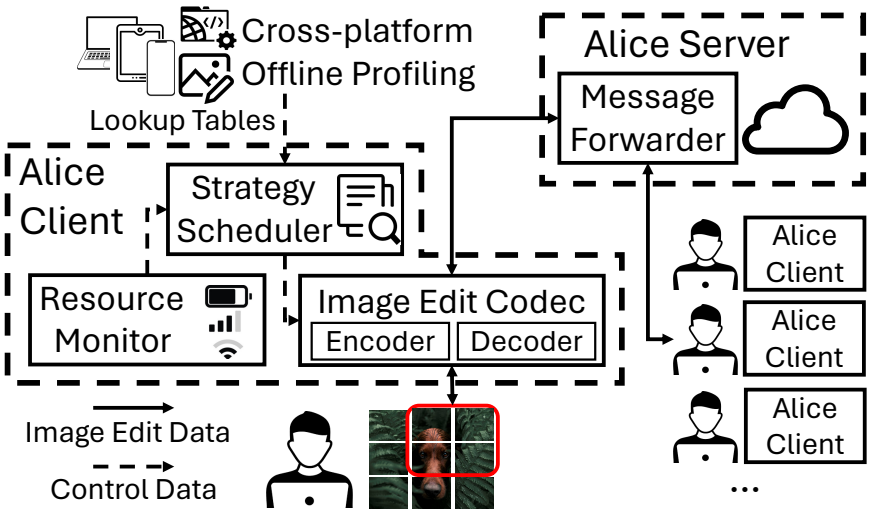


Figure 5.1: The system architecture of Alice.

5.2 Alice **Oveview**

Alice is a cross-platform adaptation framework designed for low-latency image live co-editing. It offers two key features: (1) *a combined use of data-based and operation-based image edit transmission approaches* (§5.3, §5.5.1), and (2) *real-time selection of the transmission strategy* (§5.5.2). Figure 5.1 illustrates the workflow of Alice, with most components operating on the client side. In an image LCE session, user edits are encoded using Alice 's image edit codec before being transmitted to other users, who then decode and apply them to the shared image. The strategy scheduler determines the compression configuration based on resource estimates (*i.e.,* bandwidth and computation) provided by the resource monitor. The Alice server forwards valid image edits (after conflict management) to users.

5.3 Understanding Real User's Image Editing Pattern

We conduct a user study with three objectives: (1) understand real user image editing patterns, (2) validate the tile-based image edit transmission design outlined in §2.3.1, and (3) provide additional insights for image LCE system design.

5.3.1 Methodology

The high-level methodology of our user study is to have participants complete a series of pre-defined image editing tasks on various pre-defined images. We use a professional image editing tool [3] to log users' operations and edits. Due to the complexity of implementing a professional-grade image live co-editing tool, this study focuses on analyzing the editing patterns of individual users instead. As shown in the previous section, these results remain valuable as they provide insights into the editing behavior of professional creators, which, in turn, inform the design of our Alice system. Additionally, we exclude generative AI-based image editing operations [160].

5.3.2 Results & Insights

We collect a total of 19,819 image edits performed by seven participants using 143 distinct image editing operations with diverse parameters. We then present our analysis results and insights.

Frequencies of Image Editing Operations. Figure 5.2 presents the frequencies of image editing operations, highlighting the 10 most common actions. We derive two key insights from these results. First, a significant portion of image editing operations impose minimal computational overhead on the host machine, such as Layer Visibility. This suggests that instead of relying on *data-based* transmission strategy, image LCE systems can *transmit only the operation*

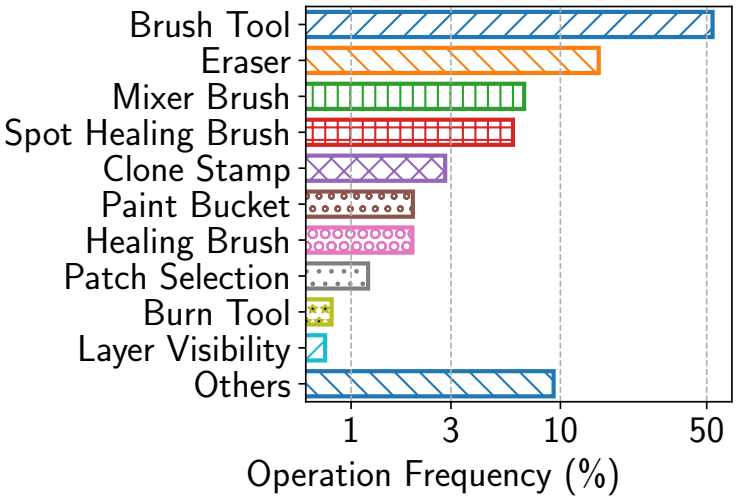


Figure 5.2: The operation frequency in our user study.

API and parameters, which typically require far less bandwidth than pixel data. Other users can then replicate the edit by replaying the operation locally with the received parameters – an approach we refer to as *operation-based* transmission. Second, certain image editing operations, such as Patch Selection, do not alter image pixels and therefore do not require transmission.

Resolutions of Image Edits. To analyze the area sizes affected by image edits, we define a set $\mathcal{S}$ consisting of 7 resolutions: $\mathcal{S} = \{128^2, 256^2, 384^2, 512^2, 1024^2, 2048^2, 4096^2\}$. The resolution of an image edit is determined as the smallest resolution in $\mathcal{S}$ that fully encompasses the edit. Our findings show that the two most common resolutions among collected image edits are 1024^2 and 384^2. In addition, the average height and width of these edits are 660×660 pixels. *These results validate the design choice in basic image LCE systems (see §2.3.1), which employs a tile-based approach for image edit transmission.*

Complexity of Image Edit Tiles. We analyze the complexity of collected image edit tiles and find that they are mathematically "simpler" than regular image tiles of the same resolution.

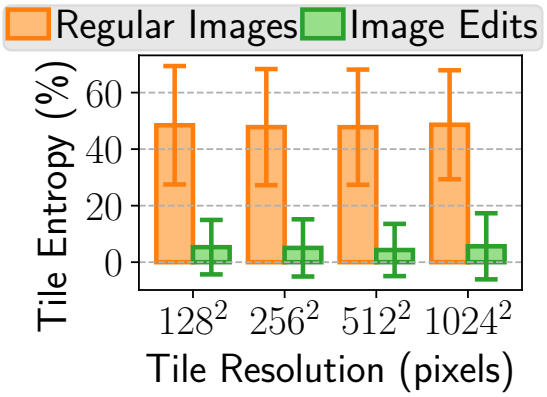


Figure 5.3: The tile entropy of regular image and image edits.

The complexity of an image edit can be quantified by its entropy [189], measured as the proportion of data size reduction after lossless compression. To investigate this, we segment image edits into tiles as defined in §2.3.1, focusing on four resolutions: $\{128^2, 256^2, 512^2, 1024^2\}$. We randomly sample 1,000 image edit tiles at each resolution and compare their complexity to an equal number of regular image tiles, sampled from our custom dataset (discussed in §5.4.1). We use PNG [174] for lossless compression. As shown in Figure 5.3, the average entropy of image edit tiles at 128^2, 256^2, 512^2, and 1024^2 is lower than that of regular image tiles at the same resolutions by 43.13%, 42.74%, 43.46%, and 43.01%, respectively. These results confirm the simplicity of image edit tiles. Our key insight is that, *given the inherent simplicity of image edit tiles, image LCE systems have ample opportunities to reduce end-to-end latency by employing less complex compression techniques or configurations. This can significantly accelerate lossless compression while maintaining the compression ratio.*

5.4 Lossless Compression For Image LCE

Many lossless compression techniques [6, 174, 196, 175, 30, 7] have been developed, yet none has been specifically investigated in the context of image live co-editing. We evaluate their performance using a comprehensive dataset we collected. We collect a separate dataset instead of reusing the one from §5.3 because we aim to investigate a broader range of image types.

5.4.1 Frameworks, Dataset & Methodology

Lossless Compression Frameworks. We explore three frameworks: a general-purpose lossless data compression framework, zlib [175], and two dedicated image compression frameworks, PNG [174] and JPEG XL [6]. The reference implementations used are zlib-1.3.1 [266], libpng-1.6.43 [112], and libjxl-0.10.0 [110]. These frameworks offer 10, 2, and 10 configurations, respectively, to balance compression ratio and efficiency. In total, we examine 20 configurations: 9 from zlib, 2 from PNG, and 9 from JPEG XL. We exclude one zlib configuration that performs no compression and one JPEG XL configuration due to excessively long compression times.

Image Tile Dataset. We construct a comprehensive image tile dataset by randomly sampling a subset from multiple popular public image datasets [95, 231, 13, 14, 84, 121, 214, 44, 98]. This results in 881 images with diverse resolutions, categories, and complexities, including photographic images, photorealistic images, artistic images, AI-generated images, and simple binary masks. Each image is then divided into tiles according to the definition in §2.3.1. We consider four tile resolutions: $\{128^2, 256^2, 512^2, 1024^2\}$. For each resolution, we get 183K+, 47K+, 12K+, and 3K+ tiles, respectively.

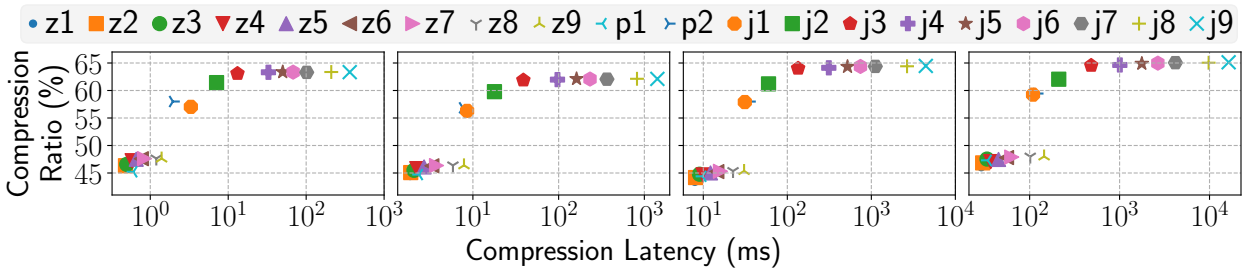


Figure 5.4: Average compression (encoding + decoding) ratio/latency of zlib (z1-9), PNG (p1-2), and JPEG XL (j1-9) under various image tile resolutions (left to right): 128^2, 256^2, 512^2, 1024^2.

Methodology. We evaluate the compression ratio and compression latency (encoding + decoding) of the above frameworks with our dataset, using a single CPU thread on a MacBook Air M1 2020 [124]. We consider both native and web (WebAssembly (Wasm) [65]) applications. Internal parallelism in compression engines is disabled, if available.

5.4.2 Compression Performance

Figure 5.4 presents the benchmark results across diverse tile resolutions. We derive four key takeaways: (1) zlib-{1-6} demonstrate a good trade-off between compression ratio and latency. However, from zlib-6 to zlib-9, improvements in compression ratio become marginal, while compression latency increases significantly; (2) The two PNG configurations exhibit notable differences in both compression ratio and latency; (3) JPEG XL-{1-3} provide a reasonable trade-off between compression ratio and latency. In contrast, JPGE XL-{4-9} incur a substantial increase in compression latency with only minimal gains in compression ratio; and (4) More importantly, *the significant heterogeneity among all configurations presents opportunities for image LCE systems to select compression configurations at runtime to adapt to varying resource constraints.*

5.5 System Design of Alice

We now present the system design of Alice, as illustrated in Figure 5.1.

5.5.1 Hybrid Transmission Strategies

Alice combines *data-based* and *operation-based* approaches for image edit transmission. In *data-based* transmission, pixel data from an image edit is sent from one user to others, with or without lossless compression. In *operation-based* transmission, Alice transmits the metadata of the operation (*e.g.,* API and parameters) from one user to others, who then replicate the edit by replaying the operation locally with the received parameters. This hybrid approach introduces a system compatibility challenge for Alice, which stems from two key issues. First, clients in a live co-editing session may have different sets of image operations due to varying application versions (*e.g.,* across operating systems) or platform resources (*e.g.,* native vs. web applications, laptop vs. tablet). Second, many image editing operations rely on specific graphic assets or presets [40], which may not be publicly available. As a result, blindly adopting the hybrid transmission strategy without precautions may lead to transmission failures due to operation incompatibility or missing graphic assets. To address this challenge, Alice enforces two policies: (1) *operation-based* transmission is restricted to common operations that are supported across all variants of the image LCE application. These operations can be negotiated at the start of an editing session between remote clients; and (2) the Alice client monitors the graphic assets used in each editing operation and switches to data-based transmission whenever private assets are involved.

5.5.2 Real-time Strategy Selection

Alice must solve a discrete optimization problem in real time to determine the optimal transmission strategy for image edits. This problem involves a large search space, including: (1) selecting between the *operation-based* and *data-based* approaches, and (2) choosing from all available configurations of the *data-based* approach. There are three additional challenges: (1) measuring the exact end-to-end latency of each configuration through actual execution is impractical, as it would require redundant transmissions of the same image edit; (2) the selection algorithm must be fast to prevent additional latency overhead; and (3) each LCE client must be aware of available resources on other clients to optimally solve the problem.

Inspired by FastMPC [248], Alice overcomes these challenges by efficiently selecting the optimal compression strategy at runtime using an LUT-based approach. Each Alice client maintains two LUTs: dataLUT for the *data-based* approach and opLUT for the *operation-based* approach. As shown in Figure 5.5, dataLUT is indexed by network bandwidth and provides the predicted optimal data-based compression configuration along with its associated compression latency and ratio. opLUT is a two-level LUT: the first level is indexed by the operation ID, which accesses a sub-LUT for each image operation. The sub-LUT is indexed by specific parameters to predict execution speed based on those parameters. At runtime, for each image edit, Alice: (1) predicts *data-based* transmission latency by querying compression latency and ratio from the dataLUT, using the latest bandwidth estimate, (2) forecasts *operation-based* transmission latency by combining operation execution speed from the opLUT, size of the operation parameters, and estimated bandwidth, and (3) selects the strategy with the lowest predicted latency for the image edit. Each {platform, tile resolution} combination has its own dataLUT and opLUT, which are constructed

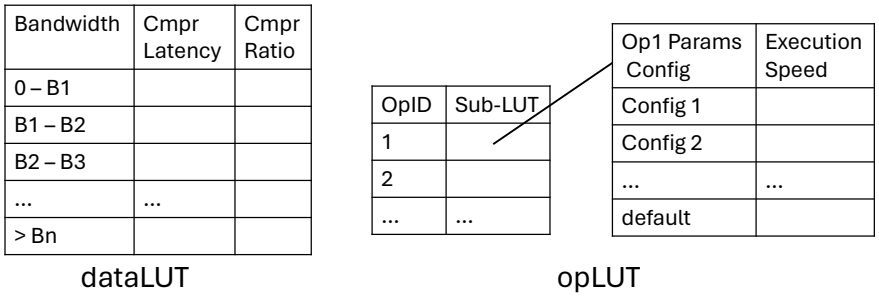


Figure 5.5: An example of the dataLUT and opLUT of Alice.

offline through extensive performance profiling and enumeration, making this a one-time setup. Next, we detail the LUT construction process.

Constructing the dataLUTs. Alice builds a dataLUT in two steps. First, it profiles the compression ratio and latency of all considered compression techniques across various configurations. This profiling follows the methodology in §5.4, using a comprehensive image tile dataset. Next, Alice enumerates possible bandwidth values and selects the compression configuration that minimizes latency for each bandwidth. The latency of each configuration is estimated using back-of-the-envelope calculations based on the average compression ratio and latency obtained in the first step. To optimize the dataLUTs, Alice applies several trimming strategies: (1) Initially, it profiles all configurations on a dataset subset before constructing the full dataLUT. It discards configurations with high compression latency and limited compression ratio improvement, such as zlib-{7-9} and JPEG XL-{7-9}, as shown in Figure 5.4; (2) Configurations that consistently perform worse than others are removed; and (3) Instead of enumerating all possible bandwidth values, Alice identifies a bandwidth range where each configuration performs optimally. Higher bandwidth requires less compression effort for image edits, making this range easily to be identified.

Building the opLUTs. The opLUTs are constructed through extensive offline measurements of various image editing operations, with two key differences from dataLUTs. First, Alice does

not enumerate bandwidth values for opLUTs. Instead, it predicts *operation-based* transmission latency at runtime. Second, image editing operations can have infinite possible parameter values, unlike the finite configurations in data-based transmission. Profiling every parameter set is impractical. To address this, Alice profiles a subset of frequently used parameters, which may include the top-k most common parameters identified from large-scale user studies. This subset is dynamically updated as Alice collects more user operation data. In addition, each operation's LUT includes a *default* entry, which provides the average execution speed across all profiled parameter sets. If a query does not match a specific entry, Alice returns this default value. We empirically set k to 20 to ensure fast table lookup, resulting in an opLUT size of ~2MB (assuming 1,000 operations and each entry takes 100 Bytes).

5.6 Implementation

Our **implementation** consists of: (1) an image LCE system following the architecture in Figure 2.2, comprising 2,182 lines of code (LoC), and (2) the Alice framework (Figure 5.1) with 2,777 LoC, integrated into the LCE system. Both components are implemented in C/C++. We develop a simple message-oriented protocol over TCP for communication between the server and clients. For *data-based* transmission, Alice applies zlib-1.3.1 [266], libpng-1.6.43 [112], and libjxl-0.10.0 [110] with configurations 1-6, 1-2, and 1-3, respectively.

5.7 Evaluation

5.7.1 Experimental Setup

Controlled Experiments. We conduct trace-driven evaluations, where each image LCE client is assigned an image editing trace, an uplink throughput trace, and a downlink throughput trace.

Image Editing Traces. Each data point in an editing trace includes operation metadata (API and parameters), capturing time, and affected pixels and their values. To generate image editing traces, we first create an operation set with 20 image processing operations from OpenCV-4.9.0 [150]. We consider 4 trace durations: 60, 120, 180, and 240 seconds, and synthesize 20 traces per duration: For a t-second trace, we first determine the total number of operations n ($n < t$), randomly sampled from our operation set. The capturing time for the n operations are uniformly sampled from $[0, t]$. Then, for each operation, we randomly select an area from an image in our dataset, apply the operation to that area, and record the modified pixels along with their values. The operation parameters are randomly generated.

Network Traces. We sample network traces from the FCC mobile broadband dataset (January 2023) [131] and replay them using Mahimahi [145]. We use 20 uplink traces and 20 downlink traces. The average uplink throughput ranges from 10.21 to 12.34 Mbps, with a standard deviation of 10.25 to 13.02 Mbps. The average downlink throughput ranges from 64.47 to 73.51 Mbps, with a standard deviation of 59.38 to 76.15 Mbps.

Devices, Dataset, and Other Configurations. We use an Ubuntu 18.04 desktop (64-GB memory, Intel Core i9-10900X CPU @ 3.70GHz) as the LCE server. LCE clients run on two difference platforms, evenly distributed: an Ubuntu 20.04 desktop (32-GB memory, Intel Core i7-9700K CPU @ 3.60GHz) and a MacBook Air 2020 laptop. We randomly select 100 images from our

Baseline	Transmission	Compression
op	*operation-based*	N/A
data	*data-based*	Dynamic configuration
raw	*data-based*	No compression
zlib3	*data-based*	zlib-3 in §5.4.1
png1	*data-based*	PNG-1 in §5.4.1
jxl1	*data-based*	JPEG XL-1 in §5.4.1

Table 5.1: Comparison baselines in our experiments.

dataset (§5.4.1) for evaluation and consider diverse *tile resolutions* (512^2 and 1024^2) and *client scales* (2, 4, 6, 8, and 10 clients).

Baselines. We consider 6 baselines, summarized in Table 5.1: (1) *op* applies only the *operation-based* approach; (2) *data* applies only the *data-based* approach with online configuration selection; and (3) 4 *data-based* baselines with fixed configuration: *raw* applies no compression, *zlib3* uses zlib configuration 3, *png1* uses PNG configuration 1, and *jxl1* uses JPEG XL configuration 1.

Metrics. We evaluate Alice in terms of the per-tile end-to-end latency and its overhead. Per-tile latency is defined as the elapsed time between when an image edit on a tile is captured and when the tile is displayed on the other client's device. We focus on per-tile latency because, in our prototype, tiles are compressed and transmitted sequentially, making per-tile latency a reasonable approximation of per-edit latency. All results are reported across all images, traces, network traces, and clients.

5.7.2 End-to-end Performance of Alice

Alice **vs. Baselines.** We first compare Alice with the baselines in Table 5.1, considering 2 LCE clients with a tile resolution of 1024^2. Figure 5.6 shows our result. We have three takeaways. First, compared to *raw*, *zlib3*, *png1*, and *jxl1*, *data* reduces the average (median) per-tile latency by

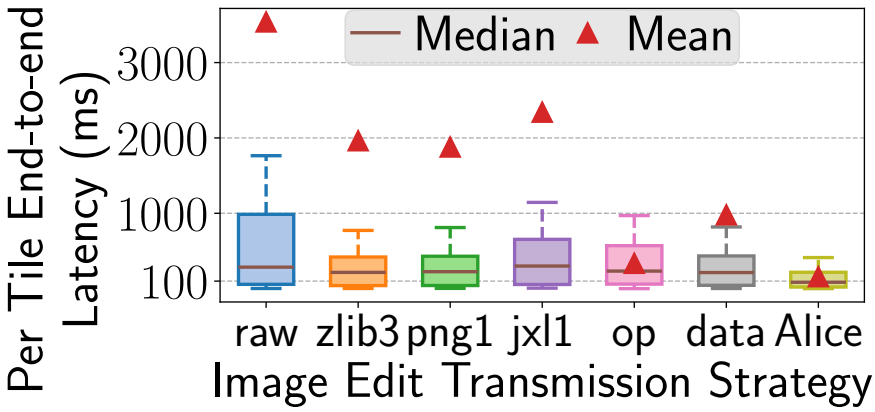


Figure 5.6: Per-tile latency of different image edit transmission strategies. Tile resolution is 1024^2. The # of clients is 2.

90.44% (49.91%), 78.16% (9.62%), 78.30% (11.27%), and 75.31% (20.42%), respectively, thanks to the dynamic compression configuration selection at runtime, which adapts to the varying bandwidth better, compared to using a fixed configuration. Second, compared to *data* and *op*, Alice further reduces the average (median) latency by 62.03% (28.34%) and 66.72% (79.09%), respectively. This confirms the effectiveness of jointly using *data-based* and *operation-based* approaches. Third, the average (median) per-tile latency of Alice is 160.18ms (85.01ms), which is a 95.48% (70.18%), 91.85% (60.55%), 91.49% (62.21%), and 93.17% (71.65%) reduction compared to *raw*, *zlib3*, *png1*, and *jxl1*, respectively.

Various Tile Resolution. We repeat the evaluation with a tile resolution of 512^2. As shown in Figure 5.7: (1) compared to *raw*, *zlib3*, *png1*, and *jxl1*, *data* reduces the average (median) per-tile latency by 90.44% (49.91%), 78.16% (9.62%), 78.30% (11.27%) and 75.31% (20.42%), respectively; (2) compared to *data* or *op*, Alice further reduces the average (median) per-tile latency by 62.03% (28.34%) and 66.72% (79.02%), respectively; and (3) the average (median) per-tile latency of Alice is 115.06ms (58.87ms). This result confirms Alice's effectiveness under various tile resolutions.

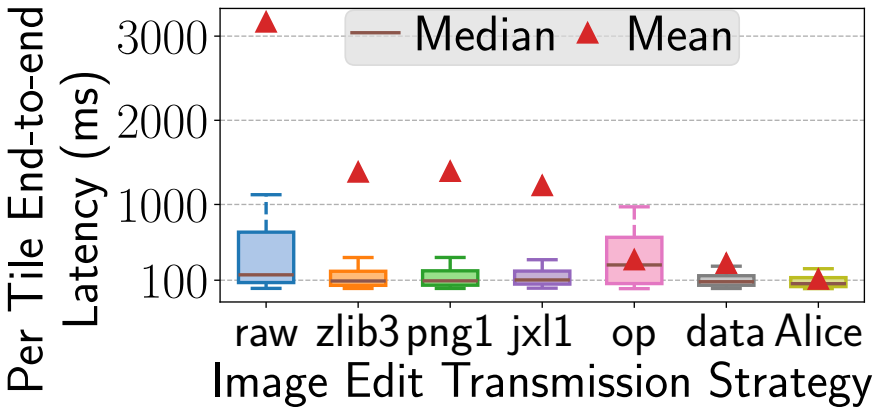


Figure 5.7: Per-tile latency of different image edit transmission strategies. Tile resolution is 512^2. The # of clients is 2.

Diverse Client Scales. We vary the number of LCE clients from 2 to 10, and compare Alice with *op* and *data* in Table 5.1. Figure 5.8 shows that when the number of clients is 2, 4, 6, 8, and 10, Alice consistently outperforms the baselines: (1) the average (median) per-tile latency of Alice is 160.18ms (85.01ms), 169.62ms (85.02ms), 155.36ms (84.52ms), 178.99ms (85.00ms), and 177.35ms (82.57ms), respectively; (2) Compared to *data*, Alice reduces the average (median) latency by 83.68% (60.13%), 82.60% (65.07%), 84.70% (62.79%), 81.01% (66.08%), and 76.40% (68.88%), respectively; (3) Compared to *op*, Alice reduces the average (median) latency by 52.59% (63.49%), 49.80% (63.71%), 54.00% (63.90%), 47.04% (63.72%), and 47.50% (64.64%), respectively. This result confirms the scalability of Alice.

5.7.3 Micro-benchmarks

LUT-based vs. ML-based Selection. We compare the LUT-based strategy selection with a machine learning-based approach (Alice-ML). Specifically, we formulate strategy selection as a classification problem, where a machine learning model predicts the optimal strategy based on: device type ID, tile resolution, operation ID, and bandwidth estimation. Using an offline-collected

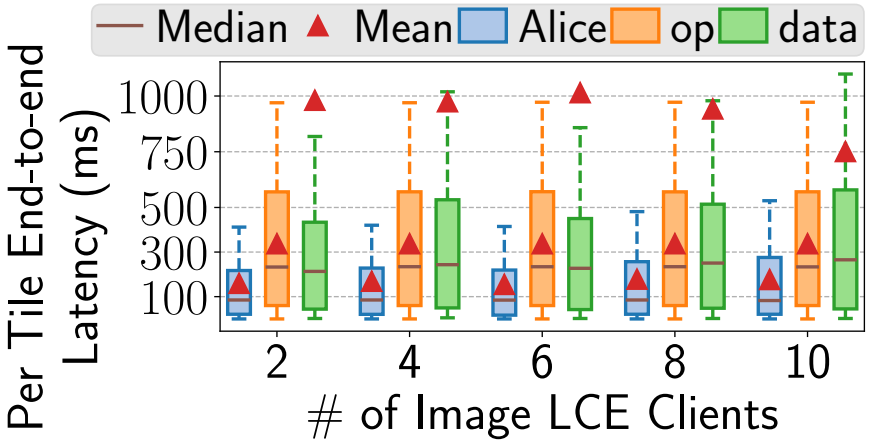


Figure 5.8: Per-tile latency with various # of clients. Tile resolution is 1024^2.

dataset, we investigate 4 lightweight ML models: *Logistic Regression* (LR), *Random Forest* (RF), *Gradient Boosting Decision Tree* (GBDT), and *Multi-layer Perceptron* (MLP). We implement these models using Python's *scikit-learn* package with default parameters. Table 5.2 shows their prediction accuracy through 10-fold cross-validation, where GBDT achieves the best performance. We then integrate the pre-trained GDBT model into Alice-ML and compare its performance with Alice. We set the tile resolution to 1024^2 and the number of clients to 2. Figure 5.9 shows that Alice outperforms Alice-ML, reducing average (median) latency by 81.83% (56.91%). The likely reason for Alice's superior performance is the limited scale of the training dataset and the lack of fine-tuning in the ML model. Nevertheless, this result demonstrates the feasibility of ML-based solutions as an alternative approach for dynamic configuration selection, which we plan to further explore in future work.

Alice **Overhead.** We confirm that Alice incurs negligible overhead (<1ms) for transmission strategy selection on our test devices.

Model	Accuracy
LR	51.64
RF	55.91
GBDT	**61.76**
MLP	52.13

Table 5.2: Prediction accuracy of explored ML models.

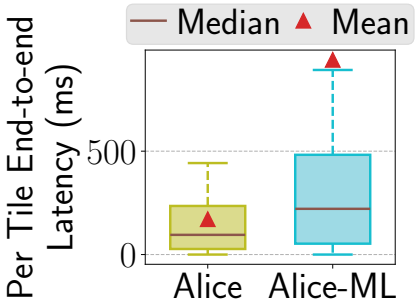


Figure 5.9: Alice vs. Alice-ML. The tile resolution is set to 1024^2.

5.8 Summary

This chapter focuses on achieving low-latency image live co-editing. We demonstrate that by jointly utilizing *data-based* and *operation-based* strategies and integrating diverse lossless compression techniques and configurations, an image LCE system can achieve up to 95% latency reduction compared to baselines. We believe that our high-level design concept can also benefit human-machine co-editing applications in the Generative AI era.

Chapter 6

NIER: Practical Neural-enhanced Low-bitrate Video Conferencing

6.1 Introduction

In this chapter, we switch to *low-bitrate video conferencing*, a networked multimedia application that is more relevant to our daily life. As mentioned in §2.4, low-bitrate video conferencing can benefit multiple stakeholders, including streaming platforms, cellular providers, mobile customers, and end users. However, the existing SR-based low-bitrate conferencing solution is suboptimal, due to the heavy temporal dependency introduced by traditional video codecs for encoding low-resolution video frames.

The objective of our work in this chapter is to develop a practical low-bitrate video conferencing solution, referred to as NIER. Our design goals are: (1) adaptively maintaining a target bitrate between 10 and 100 Kbps with reasonable video quality; (2) being robust to packet losses; and (3) being practically deployable on commodity devices wtih a frame rate of 30+ FPS. We believe

that satisfying these above requirements makes NIER suitable for a wide range of usage scenarios, including poor wireless signal conditions, metered mobile connections, cellular roaming, and in-flight Wi-Fi.

To achieve the above goal, we leverage a technique called *key-point-based deep image animation* (DIA) as a key building block. As mentioned in §2.4.1, DIA was originally designed to animate a static image using the motion and deformation of a video clip [242, 192, 227]. Recently, researchers in the computer vision community have explored the use of key-point-based DIA in low-bitrate video calls [4, 153]. In this approach, motion and deformation (*e.g.,* the optical flow [23] between two images) are "encoded" as sparse key-points (consisting of coordinates and attributes) transmitted from the sender. At the receiver side, these key-points are used alongside a high-quality "reference frame" to generate corresponding frames through a pre-trained DIA model. Compared to traditional pixel-based codecs, the key-point representation bears a much lower bitrate. In addition, since each frame is independently encoded into key-points, packet losses affect only corresponding frames, rather than propagating errors across multiple frames.

Despite its potential, existing studies [96, 4, 153] focus on improving key-point-based DIA models while overlooking critical challenges in the networking and system dimensions, which are identified by our case study in §2.4.2:

Challenge 1. *When and how to transmit a reference frame?* A reference frame serves a similar role to an I-frame in traditional codecs, as it provides a more recent (and oftentimes better) "baseline" for frame generation. However, key differences between them create unique yet underexplored optimization opportunities – in particular, when and how to transmit a reference frame. In contrast, existing approaches [4, 153] typically send a reference frame only once at the beginning, causing significant quality degradation over time.

Challenge 2. *How to adapt DIA to the fluctuating bandwidth?* This can be regarded as "Adaptive Bitrate (ABR) streaming" [25] for DIA, which is largely an uncharted territory.

Challenge 3. *How to handle packet losses?* Likewise, there lack methods allowing DIA to dynamically adapt to varying packet loss rates, limiting its robustness in the real world.

Challenge 4. *How to make DIA practical on commercial off-the-shelf (COTS) devices?* Our measurements indicate that state-of-the-art key-point-based DIA [193] exhibits poor performance on COTS devices (*e.g.,* 11 FPS with 100+ ms frame processing latency on a MacBook Air with Neural Engine [10]), making it falling far short for practical use.

To the best of our knowledge, NIER is a first practical low-bitrate video conferencing system enhanced by key-point-based DIA. NIER maintains two streams between the sender and receiver: a key-point stream and a reference stream. It achieves low bitrate by transmitting most video frames as key-points, with adaptive reference frame delivery as needed.

• To address **Challenge 1**, NIER judiciously updates the reference frame by jointly considering the bandwidth constraint and visual quality impact. A challenge here is that the visual quality groundtruth of a to-be-generated frame is unknown. We thus devise a lightweight approach to predict the visual quality by leveraging a new metric called *self-similarity*, *i.e.,* the similarity between a to-be-generated frame and the reference frame. We find that the self-similarity is highly correlated with the visual quality groundtruth and can be easily derived on the sender side, making it a good predictor. In addition, instead of discarding old reference frames, NIER opportunistically reuses them to further boost the QoE.

• To address **Challenge 2**, our key insight is that the key-point stream and the reference stream, which compete for bandwidth, exhibit distinct characteristics: the key-point stream demands low bandwidth but requires immediate delivery, whereas the reference stream consumes high

bandwidth yet remains delay-tolerant. NIER hence employs different strategies for them. For the key-point stream, it adopts a layered encoding scheme that encodes the key-point data into a base layer and three enhancement layers. At runtime, enhancement layers can be flexibly dropped to meet the bandwidth budget. For the reference stream, NIER reshapes its traffic pattern to make it less bursty and more elastic.

• To address **Challenge 3**, We make two observations: (1) similar to pixel-based videos, key-points exhibit temporal locality; and (2) using a recent reference frame with some missing pixels can oftentimes yield a higher generation quality than using an old non-corrupted reference frame. Therefore, for the key-point stream, NIER applies lightweight approaches to infer the missing key-points on the receiver side using historical data when packet loss occurs. For the reference stream, NIER reconstructs a reference frame from partially received segments with negligible overhead.

• To address **Challenge 4**, NIER applies a series of optimizations, including removing redundant computation, pruning/modifying DNN blocks, reducing input data, and pipelining processing stages. Many of these optimizations go beyond standard deep learning inference optimizations by considering the unique characteristics of key-point-based DIA.

We implement the above components and integrate them into a deployable prototype comprising 13K+ lines of code. Our extensive evaluations indicate that NIER meets all the aforementioned design goals: low bitrate, reasonable visual quality, resilience to packet loss, and high frame rates. We highlight key evaluation results as follows.

• Under ultra-low bandwidth (< 50 Kbps) with a 50 ms one-way delay, NIER achieves 205 (225) ms 50th (95th) percentile (P50 and P95) end-to-end latency, a 99.8% (99.9%) reduction compared to a baseline design where key-point-based DIA is applied in a straightforward manner according to

the computer vision literature [4, 153]. In addition, NIER improves the decodable frame ratio by 11.47% under a 10% packet loss rate and improves the visual quality (in PSNR) by 2.03 dB.

• Compared to a SOTA low-bitrate video conferencing solution [195] enhanced by super resolution, NIER improves the P50 (P95) end-to-end latency by up to 98.5% (99.1%), and achieves up to 159x improvement in decodable frame ratio, with a comparable or even better visual quality.

• Compared to a SOTA loss-resilient neural codec [36] and a SOTA FEC scheme [180] for real-time streaming, NIER exhibits much better coding efficiency, in terms of one or more metrics (processing latency, frame rate, and data usage, *etc.*).

• Our IRB-approved user trial involving 20 subjects suggests that NIER outperforms other low-bitrate video conferencing solutions, which uses H264, VP8 and super-resolution [195], by 2.0, 1.45, and 1.7 (in the scale of 1-5), respectively.

While NIER is designed for key-point-based DIA, its high-level design principles are potentially applicable to other neural-based video streaming systems that involve heterogeneous streams (*e.g.,* [195, 36]). This study does not raise ethical issues.

6.2 NIER Overview

NIER is, to our knowledge, the first practical low-bitrate video conferencing system enhanced by key-point-based DIA. Our high-level design concepts are potentially applicable to other neural-based video streaming systems involving heterogeneous streams, such as Gemino [195] and Grace [36]. As shown in Figure 6.1, it maintains two streams between the sender and the receiver: a key-point stream and a reference stream. NIER realizes low bitrate by transmitting most of the video frames as key-points (coordinates + attributes, §2.4.1), and generating the frames (motion

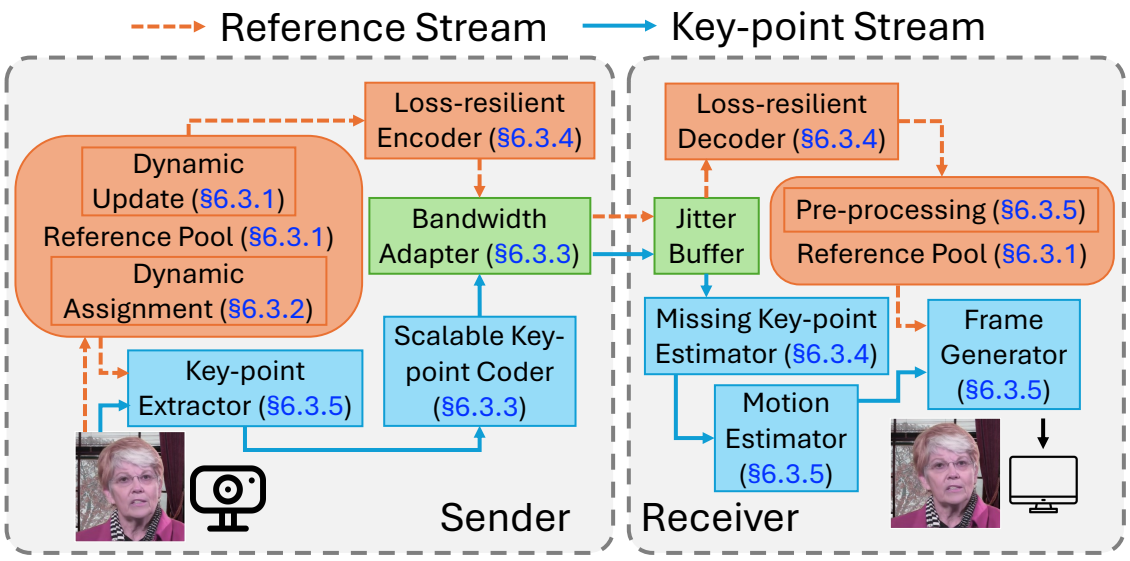


Figure 6.1: The system architecture of NIER.

estimation + frame generation) from the key-point stream along with the reference frames from the reference stream on the receiver side.

To address Issue 1 (§2.4.2), NIER dynamically updates reference frames. This incurs a crucial trade-off between the video quality and bandwidth usage, which is balanced by NIER's principled decision-making algorithm on the sender side (§6.3.1). In addition, NIER applies a synchronized reference pool on both sides with a least frequently used (LFU) based eviction policy and dynamically assigns a best reference frame to each to-be-generated frame (§6.3.2).

To address Issue 2 (§2.4.2), NIER adopts bandwidth adaptation approaches tailored to the key-point and reference stream, respectively (§6.3.3). For the *key-point stream*, we design a scalable key-point coding (SKC) scheme to transmit the key-points as a base layer and multiple enhancement layers that can be easily dropped to meet the bandwidth budget. For the *reference stream*, NIER spreads out the transmission of reference frames by reshaping the traffic pattern. In addition, NIER always prioritizes sending key-point data over reference data to ensure timely delivery of key-points.

To address Issue 3 (§2.4.2), both key-point and reference streams are designed to be *resistant to packet loss* (§6.3.4). For the *key-point stream*, NIER employs lightweight approaches to infer missing key-points on the receiver side. For the *reference stream*, NIER strategically splits a reference frame to multiple non-overlapped sub-images by downsampling, which are independently encoded. This allows frame generation to use a partially received reference frame (*i.e.,* a subset of the sub-images) under packet losses.

Last but not least, DIA in NIER *achieves line-rate (*i.e., *30+ FPS) and low latency* with little quality drop (§6.3.5), by addressing Issue 4 (§2.4.2) through three aspects: model optimization, system-level optimization, and training optimization.

6.3 System Design of NIER

We now detail the system design of NIER (Figure 6.1) that addresses the challenges we identified in §2.4.

6.3.1 Updating Reference Frames

To achieve a good quality of frame generation over time, NIER judiciously updates the reference frames. We face three practical challenges when designing the algorithm to make updating decisions. First, in a video conferencing session, it is impractical to monitor the quality of the generated frames, since they are on the receiver's end while the corresponding ground truth frames are on the sender's end. Second, we need to strategically balance the trade-off between bandwidth use and video quality. Third, there is no global view and it is difficult for NIER to make optimal decisions.

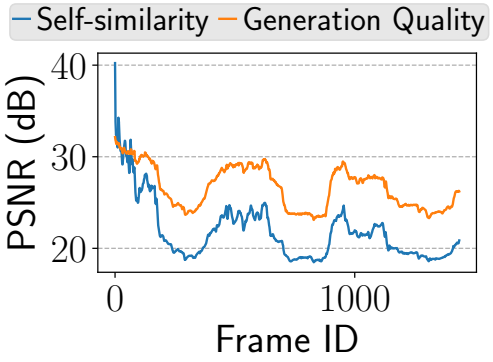


Figure 6.2: Correlation between generation quality and self-similarity: an example (reference frame is the first frame).

NIER tackles the above challenges through a principled approach that makes effective updating decisions on the sender side. Our key insight is that frame generation quality is highly correlated with the similarity between a frame and its reference frame, which we refer to as *self-similarity*. This provides NIER an opportunity to make informed updating decisions: although the quality of generated frames is difficult to track, the self-similarity of frames can easily be derived on the sender side, as elaborated next.

The Correlation between Generation Quality and Self-similarity. We first provide formal definitions of the generation quality and self-similarity of a frame f. Let the generated frame (which is generated from the key-points of f) be denoted as $\hat{f}$, and the used reference frame be denoted as f^R, the generation quality Q and self-similarity S are defined as follows, respectively:

$$Q = \mathcal{L}(\hat{f}, f) \tag{6.1}$$

$$S = \mathcal{L}(f, f^R) \tag{6.2}$$

Metric	PSNR [76]	SSIM [76]	L1 Distance [15]	LPIPS [259]
Medium	0.81, 0.78	0.79, 0.73	-0.76, -0.72	-0.80, -0.80
Average	0.78, 0.73	0.76, 0.71	-0.74, -0.72	-0.75, -0.75

Table 6.1: Medium and average values of (Pearson, Spearman) correlation coefficients between generation quality and self-similarity across 40 test videos, using different metrics.

where $\mathcal{L}(\cdot, \cdot)$ can be any metrics that reflect the pixel-level or feature-level distance between two images, such as PSNR [76], SSIM [76], L1 distance [15], and LPIPS [259]. Figure 6.2 shows an example illustrating the correlation between generation quality and self-similarity (in PSNR): when the self-similarity decreases, the generation quality drops too, and vice versa. This correlation exists because key-point-based DIA relies on the feature extracted from the reference frame f^R to generate $\hat{f}$. Therefore, a f^R that is more similar to the ground truth frame f (*i.e.,* higher self-similarity) can lead to a $\hat{f}$ that is more similar to f (*i.e.,* better generation quality).

We conduct a measurement study to quantitatively measure the correlation between the generation quality Q and self-similarity S, using 40 videos from the HDTF dataset [262] and the four different distance metrics mentioned above. For each video, we calculate the Pearson [24] and Spearman [232] correlation coefficients between the generation quality and self-similarity of all its frames, using the first frame as the reference frame. Table 6.1 shows the medium and average values of (Pearson, Spearman) correlation coefficients across all our investigated videos, using four different metrics, respectively. Note that for PSNR and SSIM, a higher value means two images are closer to each other, while for L1 distance and LPIPS, lower means closer. This explains the positive coefficient values when using PSNR/SSIM, and the negative coefficient values when using L1 distance/LPIPS. As shown in Table 6.1, the generation quality is indeed highly correlated with the self-similarity, achieving high medium and average values of the (Pearson, Spearman) correlation coefficients: (0.81, 0.78) for PSNR, (0.79, 0.73) for SSIM, (-0.76, -0.72) for L1 distance,

and (-0.80, -0.80) for LPIPS, in terms of medium; and (0.78, 0.73) for PSNR, (0.76, 0.71) for SSIM, (-0.74, -0.72) for L1 distance, and (-0.75, -0.75) for LPIPS, in terms of average.

Making Updating Decisions. We develop a lightweight algorithm for NIER to make updating decisions on the sender side, by leveraging the above correlation. It considers both the visual quality and bandwidth budget. To take the bandwidth budget into account, NIER uses a reference frame update interval of r seconds as the minimum waiting time between two consecutive reference frame updates (*Condition 1*). When updating a new reference frame, NIER dynamically adjusts r as $r = \frac{\overline{size}}{BW_{est} - BW_{kp}}$, where $\overline{size}$ is the average encoded size of the old reference frames, BW_{est} is the current bandwidth estimation, and BW_{kp} is the bandwidth reserved for streaming key-points. When $BW_{est} \leq BW_{kp}$, NIER stops updating reference frames until the bandwidth permits. To consider visual quality, NIER tracks the harmonic mean [49] of self-similarity of the previous frames within a time window of h seconds (denoted as $\overline{S}$). It then compares the current self-similarity S with $\overline{S}$, and an update is allowed if $S < \alpha \times \overline{S}$ (*Condition 2*). NIER finally updates a reference frame when both *Condition 1* and *Condition 2* are met. r is initialized to 5 seconds for cold start, and we empirically confirm that $h = 5$ seconds and $\alpha = 0.98$ work practically well.

6.3.2 Dynamic Reference Frame Assignment

In video conferencing using traditional video codecs, the IDR-frame [200], a special type of I-frames, is used to refresh the decoder state by removing any previous reference frames. Intuitively, we expect that the reference frame in the context of NIER functions similarly to the IDF-frame, *i.e.,* the old reference frame is discarded when receiving a new reference frame. However,

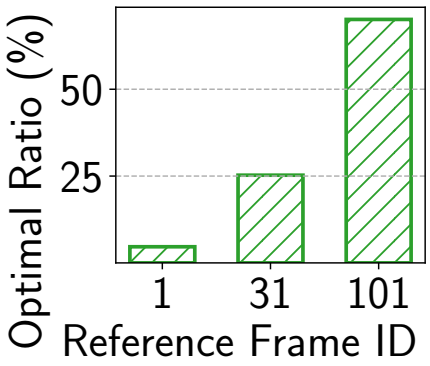


Figure 6.3: The ratios of Frame {1, 31, 101} being the optimal reference in generating Frame 102-601. The video and DIA model are from our case study (§2.4.2).

we observe that old reference frames can still be useful to improve frame generation quality, despite a more recent reference frame. We use the video and DIA model in our case study (§2.4.2) to demonstrate this finding. We first randomly pick three frames: Frame {1, 31, 101}, and then generate 500 frames starting from Frame 102. We repeat the generation three times, each time using one of Frame {1, 31, 101} as the reference frame, respectively. This allows us to identify the optimal reference frame among our three candidates that yields the highest quality (in PSNR) for each generated frame. Figure 6.3 shows the ratios of Frame {1, 31, 101} being the optimal reference frame. There are two observations. First, a more recent reference frame tends to achieve a higher generation quality more frequently, confirming the necessity of updating the reference frames. Second, the old reference frames still have a considerable amount of time, *e.g.,* more than 25% for Frame 31 in Figure 6.3, being the optimal one. We have similar findings using different videos and DIA frameworks. The second observation provides NIER an opportunity to further enhance the frame generation quality by strategically reusing the old reference frames.

Reference Pool for Dynamic Assignment. To reuse the old reference frames, NIER maintains a synchronized reference pool on both the sender and receiver side. The reference pool consists of multiple reference frames indexed by unique IDs, and its synchronization is realized

by exchanging control messages between the sender and the receiver (*e.g.,* using RTCP [83]). Recall that the generation quality is highly correlated with the self-similarity (§6.3.1). NIER thus assigns the reference frame in the pool that achieves the highest self-similarity to each frame encoded as key-points on the sender side. The ID of the reference frame and key-points are packetized together and sent to the receiver. In the worst case where the receiver cannot find the reference frame in its pool with the received ID, the most recent reference frame will be used. To speed up the sender-side assignment, NIER downsamples all the reference frames in its pool to a resolution of 64×64 for self-similarity calculation. We confirm that this has negligible impacts on the assignment.

Reference Pool Maintenance. To prevent a continuous increase of the execution time of the above algorithm, NIER sets a maximum size for its reference pool and adopts a least frequently used (LFU) based eviction policy when the pool is full and there is a new reference frame. If multiple reference frames have the same access frequency, they will be evicted in a first-in-first-out (FIFO) manner. We empirically set the upper bound of the reference pool size to 20 in NIER.

6.3.3 Adapting to Fluctuating Bandwidth

The primary challenge NIER faces in bandwidth adaptation is that, the key-point and reference streams compete the bandwidth resource with each other. Our insight is that these two streams exhibit distinct characteristics: the key-point stream requires a constant and low bitrate, *e.g.,* 56.25 Kbps for 30-FPS playback (§2.4.2), and demands immediate delivery. In contrast, the reference stream is intermittent and bears a more bursty traffic pattern. For instance, in our case study video (§2.4.2), a 512×512 reference frame, after encoding with H264, can be 31432 bytes –

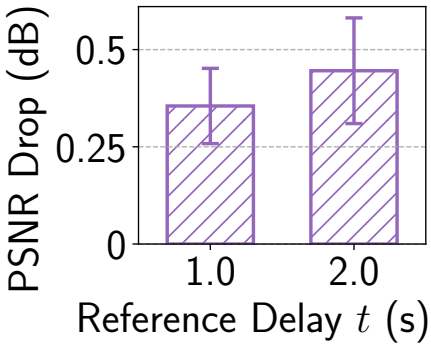


Figure 6.4: Generation quality drop under $t = \{1, 2\}$-second delayed reception of the reference frame, compared to no delays. Setup: 13 videos in §6.5.1, fixed reference update interval (every 100 frames).

130× larger than the key-point size (240 bytes per frame). In addition, the delivery of the reference frame can tolerate some delays: NIER can continue using the previous reference frame to generate new frames without causing any video freezes, even if the new reference frame is not immediately ready. To validate this, we apply our trained DIA model in §2.4.2 to the other test videos in §6.5.1, where we set a fixed reference update interval (every 100 frames) and manually delay the reception of the reference frame by $t = \{1, 2\}$ seconds. Figure 6.4 plots the average generation quality drop under different delays, compared to the ideal case where the reference frame is always received immediately. As shown, the average quality drop is indeed marginal. Considering their unique natures, NIER tailors its bandwidth adaptation solutions to the key-point and reference stream separately, and adopts a centralized scheduling algorithm that always prioritizes key-point packets over reference packets.

Scalable Key-point Coding. We develop a scalable key-point coding scheme to achieve bandwidth adaptation for the key-points. This is a similar concept as the scalable video coding (SVC), where a frame is encoded to a base layer and multiple enhancement layers (that can be dropped when bandwidth is not sufficient). Recall that a key-point p_j consists of its coordinate

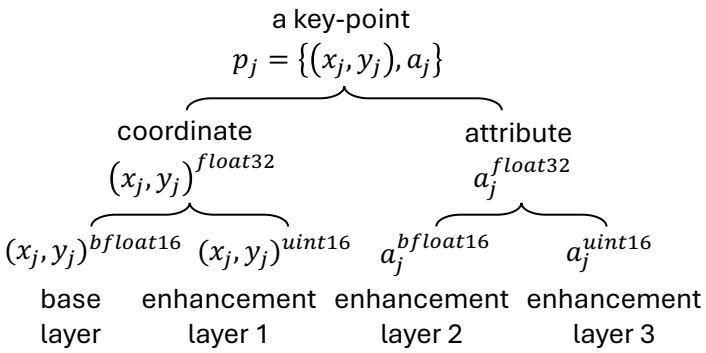


Figure 6.5: Scalable key-point coding in NIER.

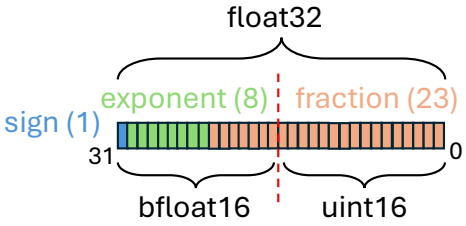


Figure 6.6: Truncating a float32 to a bfloat32 and an uint16.

(x_j, y_j) and some point-wise attributes a_j (§2.4.1). A potential way to realize scalable coding is to divide the key-points to small groups, each corresponding to a layer. However, this approach does not work since all key-points are required for frame generation. We further find that although all key-points are required, only their coordinates are indispensable for frame generation. Their attributes can be dropped with a compromised generation quality. NIER thus performs the scalable coding on a per key-point basis. As shown in Figure 6.5, NIER separates each key-point using a "top-down" approach: a key-point is first separated to coordinate and attribute, represented in single-precision floating-point numbers [122] (denoted as *float32*). Then, both the coordinate and attribute are divided into two parts, by truncating the each *float32* number to a brain floating-point number [209] (denoted as *bfloat16*) and a 16-bit unsigned integer (denoted as *uint16*), as shown in Figure 6.6. With this approach, NIER encodes the key-points of a frame to four layers as shown in Figure 6.5. Figure 6.7 shows the rate distortion (in PSNR) curve of our scalable key-point coding scheme. As shown, each enhancement layer marginally improves the generation quality while incurring non-trivial bitrate overhead compared to the base layer. This allows NIER to drop the enhancement layers (in the order of layer 3 to 1) to meet the bandwidth budget

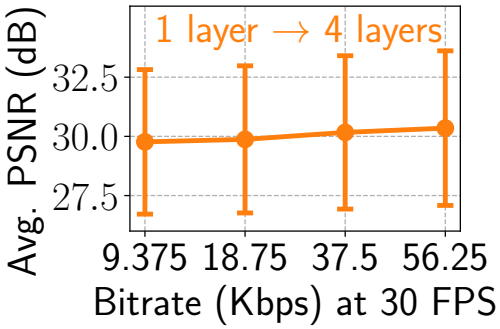


Figure 6.7: Rate distortion curve of scalable key-point coding. Setup: first 500 frames from 13 videos in §6.5.1, 10 key-points (coordinate + a 2×2 local affine transformation matrix), 30 FPS.

when needed. We also train a motion estimator (§2.4.1) that takes only the key-point coordinates as its input for key-point-based DIA.

Reshaping the Traffic for Reference Stream. For the reference stream, we have two design principles: (1) keep the reference frame in a high quality, and (2) fully utilize the bandwidth budget. Note that we keep the reference frame in a high quality because its quality can affect the generation quality of all the future frames referring to it. Aggressively lowering a reference frame's quality can make it useless, *i.e.,* leading to an even worse generation quality compared to using an old but high-quality reference frame. NIER thus does not aggressively encode reference frames to reduce the data size. Instead, NIER reshapes the traffic pattern of the reference stream by spreading out each reference frame's transmission, bringing certain delays to their delivery. This is achieved by paced-sending reference frame packets, using a leaky bucket algorithm [146] (details are in Algorithm 6.2).

Priority-based Scheduling. To guarantee the timely delivery of key-points while not adding too much delay to the reference frame, NIER (1) prioritizes sending key-point base layer packets over reference packets, and (2) drops all the key-point enhancement layer packets when there is a pending reference frame. NIER drops the enhancement layers because they only slightly affect

Table 6.2: Detailed Pseudo-Code for NIER's Bandwidth Adaptation and Packet Scheduling Logic described in §6.3.3.

Variables: $queue_{kp}$: FIFO queue for key-point packets; $queue_{ref}$: FIFO queue for reference packets; bw: bandwidth estimation in bytes per second; $interval$: scheduling interval in second; $lastRun$: last scheduling time; $budget$: budget in bytes;

```
1  Worker Thread PacketScheduler():
2      while not quit do
3          curRun ← GetCurrentTimeInSecond()
4          newBudget ← budget + (curRun − lastRun) × bw
5          maxBudget ← interval × bw
6          budget ← min(newBudget, maxBudget)
7          while not queue_kp.Empty() and budget > 0 do
8              pkt ← queue_kp.Dequeue()
9              if IsBaseLayerPacket(pkt) or queue_ref.Empty() then
10                 TransportLayerSendPacket(pkt)
11                 budget ← budget − pkt.Size()
12             else
13                 drop pkt
14         while not queue_ref.Empty() and budget > 0 do
15             pkt ← queue_ref.Dequeue()
16             TransportLayerSendPacket(pkt)
17             budget ← budget − pkt.Size()
18         lastRun ← curRun
19         SleepFor(interval);
```

the generation quality of their corresponding frames, *i.e.,* yielding a lower utility than delivering a new reference frame. Specifically, NIER uses a centralized packet scheduler with separate FIFO queues for key-point and reference streams. NIER first determines a sending budget based on the current bandwidth estimation and a sending window. It then prioritizes clearing (either sending or droping) the key-point packet queue as far as the budget permits. Any remaining budget is allocated to reference packets. More details can be found in Algorithm 6.2. This scheduling algorithm also applies to other media such as audio, which has a higher priority than key-points.

6.3.4 Enhancing Resilience to Packet Loss

There are two common packet loss recovery solutions [48]: retransmission and forward error correction (FEC). Retransmission incurs substantial extra latency, and FEC assumes a precise packet loss rate estimation. In addition, both require transmitting redundant data. While they remain applicable to NIER, they incur additional delays and bandwidth overhead. NIER instead employs additional optimizations specific to key-point and reference streams, as elaborated next.

Inferring Missing Key-points. Recall that the key-points of a frame are encoded to a base layer and three enhancement layers through NIER's scalable key-point coding scheme (§6.3.3). Each layer consumes one packet to transmit due to the small data size. NIER infers the missing key-point packet(s) on the receiver side using lightweight approaches based on the historical data. Specifically, when coordinate layers are missing, NIER linearly extrapolates the coordinate from the historical trajectory. When attribute layers are lost, NIER simply reuses the most recent attribute.

Loss-resilient Reference Frame Split. NIER encodes the reference frames using traditional video (image) codecs, and the encoded data cannot be properly decoded when some packets are lost. Our insights are: (1) a reference frame where some details are not in high resolution can still be applied to frame generation, and (2) in the worst case, NIER still functions well (thanks to the old reference frames) even if a reference frame is fully corrupted. To allow partial recovery of received reference frames, NIER splits a reference frame into multiple non-overlapped sub-images by downsampling. To split a reference frame with a size of $h \times w$ into n ($n = \frac{h}{h_0} \times \frac{w}{w_0}$) sub-images with an equal size of $h_0 \times w_0$, the reference frame is first spatially segmented into $h_0 \times w_0$ blocks, each consisting of n pixels with their pre-defined indices. The sub-image i is derived by sampling

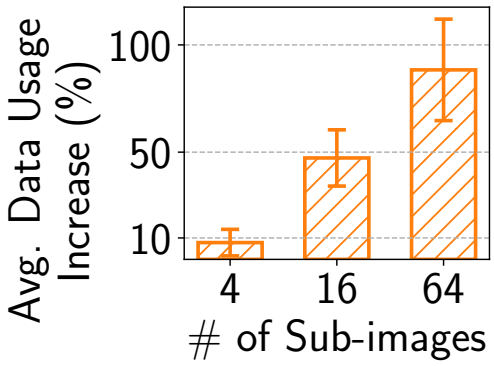


Figure 6.8: Data usage of reference split over no split.

the pixel whose index is i from each block. These sub-images are then encoded independently and sent back-to-back. The receiver reconstructs the original reference frame by reversing the split process. When the reference frame is partially received (*i.e.,* some sub-images are missing), NIER applies a 2D Gaussian filter [29] to fill out the missing pixels using their neighbor pixels in the recovered image.

A constraint NIER must consider is the extra encoded size of the reference frame, due to its split-then-encoding strategy. To demonstrate it, we randomly sample 1,000 frames (in 512×512 resolution) from our test videos (§6.5.1). We use H264 as our codec and consider splitting a reference frame into {4, 16, 64} sub-images. Figure 6.8 shows the additional data usage of our split-then-coding strategy. We empirically set the number of sub-images to 4 due to its low bandwidth overhead as shown in the figure. Note that we are aware of recent advances in loss-resilient neural codecs such as GRACE [36], which are orthogonal to our approach. We do not apply these neural codecs in NIER due to their considerable computation overhead (see Figure 6.16), which also competes resources with NIER's key-point-based DIA.

6.3.5 Accelerating Key-point-based DIA

We now describe model-level and system-level optimizations introduced in NIER.

Model Optimization. NIER accelerates the key-point-based DIA through a "top-down" approach: first optimizing the framework as a whole, and then fine-tuning each component. Recall that a key-point-based DIA framework typically consists of three DNNs: a key-point extractor, a motion estimator, and a frame generator (§2.4.1). We find that for the motion estimator and the frame generator, there are redundant operations that do not need to be executed every time. They include the feature encoding in frame generator and the reference frame pre-processing operations in motion estimation. NIER separates these redundant operations from the motion estimator and frame generator, and integrate them as a pre-processor, which is only executed when a reference frame is received.

Next, we modifiy each DNN in the framework, respectively. *To speed up the key-point extractor*, we reduce the maximum feature map size in convolution to cut down the computation overhead. In addition, we replace the 3-channel RGB image with its grayscale version as the model's input. *To accelerate the motion estimator*, in addition to applying the same optimizations we do for the key-point extractor, we also reduce the expansion ratio of feature map size for each convolution layer. Furthermore, we replace the traditional residual block [72] with the more computationally efficient inverted residual block [184]. we employ similar approaches: reducing the feature map size and its expansion ratio, and applying inverted residual blocks. Our optimizations reduce the processing latency per frame by ~52% and increase the frame rate by ~130% (see Figure 6.27).

System-level Optimization. NIER further pipelines motion estimation and frame generation, boosting the frame rate above 30 FPS with a marginal latency increase. The overall processing latency remains low (~66 ms on an Apple MacBook Air M1 2020 as shown in Figure 6.27).

Training Optimization. We also notice that our model optimization slightly degrades the frame generation quality. We add a reconstruction loss (L1 loss [15]) when training our DNNs. This helps compensate for the generation quality drop incurred by our model optimization, without causing extra inference overhead.

6.4 Implementation

We integrate all the components in §6.3.5 into NIER, a holistic system as shown in Figure 6.1. Our implementation consists of 13K+ lines of code (LoC) in C/C++. For offline key-point-based DIA model training, we implement upon the source code of FOMM [193] using Pytorch [157]. Our pre-trained models are converted to the ONNX format [148] for cross-platform execution. NIER's prototype consists of the NIER player and the NIER codec, as elaborated next.

The NIER Player. For NIER's player, we begin by developing a basic video conferencing prototype following WebRTC [120], an open-source framework that enables video and audio conferencing atop the real-time transport protocol (RTP) over UDP. We use libdatachannel [109] as the WebRTC network library, which consists of WebSockets, WebRTC Media Transport, and WebRTC Data Channels in our player for peer connection establishment, RTP/RTCP packet encapsulation, and underlying data transmission over UDP and SCTP [199] (WebRTC Data Channels), respectively. We implement all the upper layer components on our own, including (1) media data processing components for media (video and audio) capture, encoding/decoding, (de-)packetization,

and rendering; (2) a packet scheduler that paced-sends RTP packets using a leaky bucket algorithm [146]; (3) a jitter buffer for reassembling media data packets; (4) a rate controller that adopts Google Congestion Control (GCC) [32] and Receiver Estimated Maximum Bitrate (REMB) [158]; and (5) a NACK generator and handler for missing packets retransmission. For video and audio capture, we use OpenCV 4.9.0 [149] and libsoundio [113], respectively. For traditional video encoding and decoding, we integrate both H264 [200] and VPX [20]. For audio encoding and decoding, we apply Opus [152]. We employ SDL3 [188] to render video and audio.

The NIER Codec. We then implement a standalone NIER codec consisting of most of NIER's functionalities: reference update, reference assignment with reference pool, reference frame splitand reassembling, scalable key-point coding, missing key-point reconstruction, and encoding and decoding by key-point-based DIA. The pre-trained DNN models are loaded and executed using onnxruntime 1.17.0 [148]. We integrate the NIER codec into our player: (1) the key-point and reference streams are transmitted using two independent WebRTC media tracks; (2) we implement a packetizer tailored for NIER's key-point data; (3) we modify the packet scheduler to incorporate NIER's bandwidth adaptation algorithm; and (4) we use the WebRTC Data Channel for exchanging reference pool synchronization messages. Given the small engineering efforts above, we believe that NIER can be easily integrated into the official WebRTC library [56].

6.5 Evaluation

6.5.1 Experimental Setup

Videos. We use the HDTF dataset [262], which contains ~16 hours of 720p-1080p videos with audio. Each video features an upper-torso speaker against a static background. We randomly

select 45 videos of different subjects, and split them into training and test sets with a 7:3 ratio (*i.e.,* 32 for training, and 13 for test). The target resolution is set to 512×512.

Devices. We use two types of commodity devices: (1) 2× Ubuntu desktops with an NVIDIA 2080Ti GPU and 32GB memory, and (2) 2× MacBook Air M1 2020 [10] laptops with Neural Engine and 16GB memory. They represent mid-range PCs and laptops. The ICE [178] signaling server runs on a separate Ubuntu desktop for peer connection setup. The access point is a TP-Link Archer A7 [211].

Network Conditions. We consider two types of network conditions: (1) stable bandwidth with a constant packet loss rate; and (2) fluctuating bandwidth with a constant packet loss rate. Since NIER is designed for low-bitrate use cases, we adopt four stable bandwidths—105, 75, 45, and 15 Kbps—following Gemino's setup [195]. For varying bandwidth scenarios, we scale 8 broadband traces from FCC (Jan 2023) [131] to the range of $[15, 105]$ Kbps. We evaluate six packet loss rates: 0, 0.05, 0.1, 1, 5, and 10%. The network emulation uses a one-way delay of 50 ms and a packet queue size of 100.

Controlled Experiments. We evaluate NIER through controlled experiments. Specifically, we setup bi-directional 1-on-1 video calls between two identical devices (either both desktops or both laptops). Each device acts as both sender and receiver, transmitting video frames at 30 FPS over an emulated network. Packet retransmission is disabled. For network emulation, we use Mahimahi [145] on Ubuntu and DummyNet [173] with Packet Filter (PF) [130] on MacOS. Unless stated otherwise, results include all test videos and devices.

Metrics. We evaluate NIER using five key metrics: (1) *visual quality* is measured by PSNR [76] and LPIPS [259], which capture pixel-level and feature-level similarity, respectively (higher PSNR and lower LPIPS is better); (2) *end-to-end frame latency* is assessed at the 50th and 95th percentiles

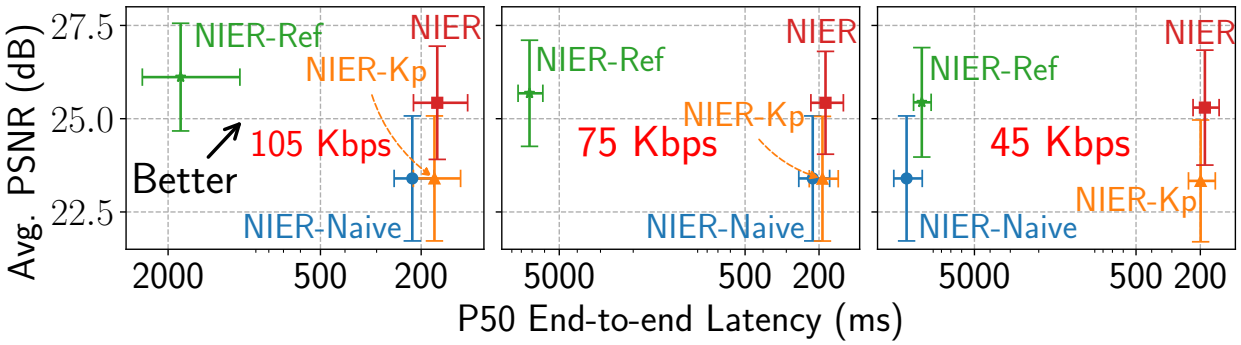


Figure 6.9: PSNR and P50 end-to-end latency of 4 NIER variants.

(P50 and P95), representing the time from frame capture at the sender to rendering at the receiver; (3) *decodable frame ratio* under various packet loss rates; (4) *video stall*, defined as an inter-frame gap exceeding 200 ms, following the industry convention [125]. We report the stall time ratio over the total video length; and (5) *encoding/decoding frame rate*.

6.5.2 End-to-end Performance of NIER

We evaluate the end-to-end performance of NIER across four variants using the above metrics. (1) ***NIER-Naive*** follows the naive system design in §2.4.2 and uses our optimized key-point-based DIA. (2) ***NIER-Kp*** builds on NIER-*Naive* by adding scalable key-point coding and missing key-point reconstruction. (3) ***NIER-Ref*** extends NIER-*Kp* by enabling the reference stream but omits priority-based packet scheduling. (4) ***NIER*** is our complete system, integrating all optimizations.

Visual Quality & End-to-end Latency. We first evaluate the visual quality and end-to-end latency of the four NIER variants under stable bandwidths of 105, 75, and 45 Kbps and zero packet loss. As shown in Figure 6.9, compared to NIER-*Naive*, NIER-*Kp* reduces P50 end-to-end latency by 18.3%, 11.3%, and 98.5% for each bandwidth while maintaining comparable PSNR. This

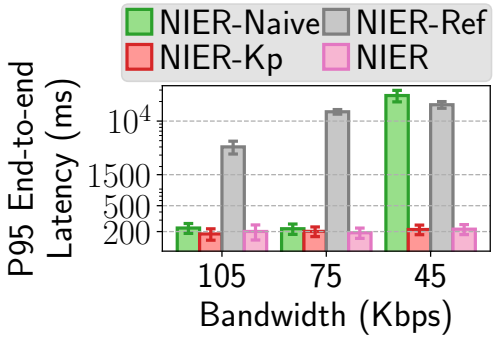


Figure 6.10: P95 end-to-end latency of NIER variants.

is achieved through scalable key-point coding scheme that allows us to adaptively drop enhancement layers to meet bandwidth budgets. Compared to NIER-*Kp*, NIER-*Ref* improves PSNR by 2.72, 2.29, and 2.11 dB, respectively, due to dynamic reference frame update and assignment. However, the PSNR gains slightly drop as bandwidth decreases because NIER-*Ref* adjusts its update frequency to meet the bandwidth budget. NIER-*Ref* also incurs significantly higher end-to-end latency due to the competition between the reference and key-point stream for limited bandwidth, delaying key-point delivery. Our full-fledged NIER significantly reduces P50 latency compared to NIER-*Ref*—90.3%, 97.4%, and 98.2%—while maintaining similar visual quality. This is due to NIER 's priority-based scheduler, which prioritizes key-point packets over reference frames. Meanwhile, NIER also improves PSNR by 2.04, 2.03, and 1.96 dB over NIER-*Kp*. The PSNR increase over NIER-*Kp* drops compared to NIER-*Ref* due to the delayed delivery of reference frames. Figure 6.10 shows that P95 latency follows the same trend as P50. We exclude results for 15 Kbps since both NIER-*Ref* and NIER revert to NIER-*Kp* at such low bandwidths, resulting in similar latency and visual quality.

Latency Breakdown of NIER**.** Figure 6.11 shows the latency breakdown of NIER on a Macbook Air M1, excluding in-network transmission time. The bars represent pipelined system components. NIER achieves 33 FPS on this commodity laptop. When the network is not a bottleneck,

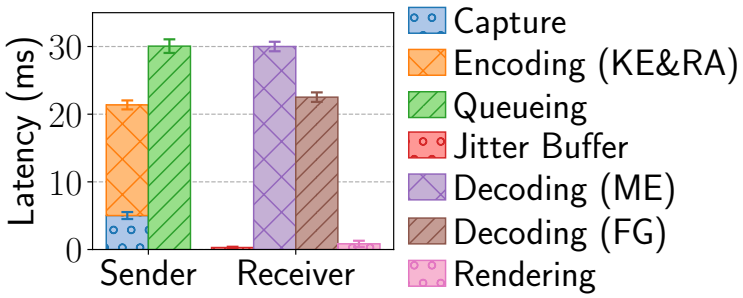


Figure 6.11: End-to-end latency breakdown of NIER. Components are pipelined. KE: Key-point Extraction; RA: Reference Assignment; ME: Motion Estimation; FG: Frame Generation.

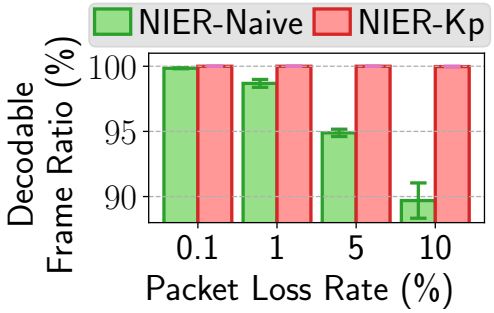


Figure 6.12: Decodable frame ratio under packet losses, at 105 Kbps.

the frame rate is primarily constrained by our DIA model's motion estimation, which takes an average 30 ms per frame. Since NIER-*Naive*, NIER-*Kp*, and NIER-*Ref* use the same DIA model, they achieve identical encoding/decoding frame rates.

Decoable Frame Ratio under Packet losses. Figure 6.12 shows the decodable frame ratio of NIER-*Naive* and NIER-*Kp* at 105 Kbps under varying packet losses. As packet loss increases from 0.1% to 10%, NIER-*Kp* maintains a decodable frame ratio above 99.98%+ thanks to scalable key-point coding (SKC) and missing key-point reconstruction. In contrast, NIER-*Naive* experiences a decline from 99.84% to 89.69%. Since NIER-*Ref* and NIER also incorporate SKC and missing key-point reconstruction, they achieve similar decodable frame ratios as NIER-*Kp*. We observe the same trend across other bandwidth conditions.

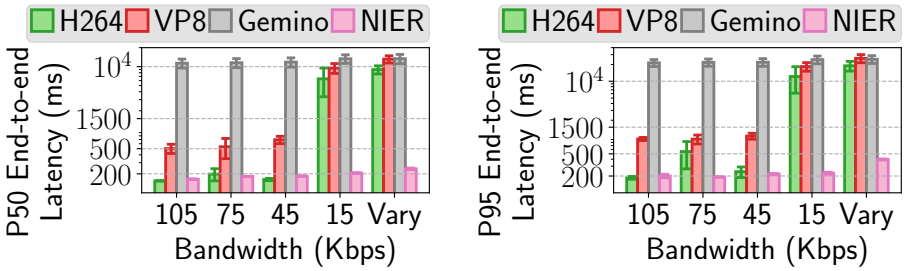


Figure 6.13: P50 and P95 end-to-end latency under stable (105, 75, 45,15 Kbps) and varying bandwidth (Vary).

6.5.3 NIER vs. Low-bitrate Schemes

We next compare NIER's performance—including end-to-end latency, decodable frame ratio, visual quality, video stall under packet loss (starting at 0.05%, a common rate in wild [180]), and encoding/decoding frame rate—against three low-bitrate video conferencing solutions: **(1) *H264***, which uses the traditional H264 [172] codec for video coding. When *H264* cannot encode the original video within the target bitrate, we downsample the frames before encoding and upsample them via bilinear interpolation [93] after decoding. Specifically, frames are downsampled to 128×128 for bitrates below 50 Kbps and to 256×256 for bitrates between 50 and 90 Kbps. **(2) *VP8***, which applies another traditional codec, VP8 [20], for video coding. It uses the same adaptation policy as *H264*; and **(3) *Gemino*** [195], a SOTA low-bitrate video conferencing solution enhanced by super-resolution (SR). *Gemino* sends a high resolution reference frame to the receiver at the start of the call, and then sends low resolution frames via traditional video codec. The *Gemino* receiver applies SR to boost the visual quality of the low-resolution frames, along with the reference frame. Since the author does not open source their code, we replicate and train a 16x Gemino model (*i.e.,* upsample a 128×128 frame to 512×512) on our own, using the dataset in §6.5.1. We use H264 [172] for encoding low-resolution frames. The trained model is converted to the ONNX format [148] and integrated into our NIER player (§6.4).

End-to-end Latency. Figure 6.13 presents the P50 and P95 end-to-end latency of NIER and our comparison baselines. We set the packet loss rate to 0 and use 5 different bandwidth traces: 4 stable bandwidth at {105, 75, 45, 15} Kbps, and our 8 traces with varying bandwidth (denoted as Vary) in the range of [15, 105] Kbps. As shown, among these 4 schemes, NIER consistently achieves the lowest end-to-end latency across all the bandwidth conditions. Even when the bandwidth is extremely low at 15 Kbps, NIER still achieves 205 (225) ms for the P50 (P95) end-to-end latency, a 96.8% (98.1%), 97.8% (98.8%), and 98.5% (99.1%) reduction compared to *H264*, *VP8*, and *Gemino* respectively. NIER's latency increases under varying bandwidth because NIER makes its reference updating decision based on the current bandwidth estimation and does not consider future bandwidth change. Meanwhile, *Gemino* consistently incurs a high end-to-end latency. The reason is two-fold. First, *Gemino*'s SR suffers from high computation overhead and can only achieve ~24 FPS, as shown in Figure 6.16. Therefore, when the sender sends the video at a constant 30 FPS, the frames pile up on the receiver side, waiting for SR. Second, even the entire video can be encoded to meet the target bitrate on average, we observe the encoded sizes of the low-resolution frames varies (as opposed to the constant key-point size per frame in NIER). As a result, a large size frame can delay the delivery of a bunch of the following frames, especially when the bandwidth is extremely low. For *H264* and *VP8*, the end-to-end latency is low when the bandwidth is sufficient. When bandwidth decreases, the end-to-end latency builds up due to the bursty encoded video traffic produced by these traditional codecs (the same reason as mentioned above for *Gemino*). *VPX* has a higher end-to-end latency than *H264* due to its more bursty traffic. Note that at 75 Kbps, the end-to-end latency of *H264* is higher than that at 45 Kbps. This is because we apply different downsampling ratios before encoding to meet the bandwidth budget (4x at 75Kbps vs. 16x at 45Kbps).

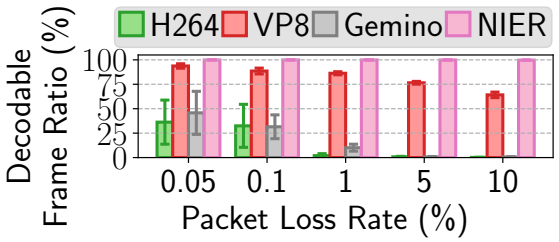


Figure 6.14: Decodable frame ratio under packet losses, at 105 Kbps.

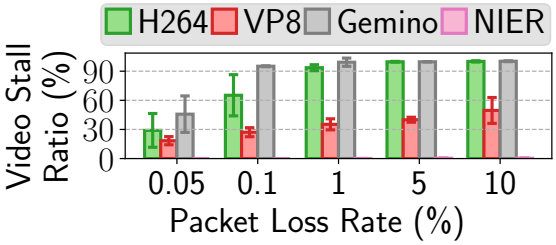


Figure 6.15: Video stall ratio under packet losses, at 15 Kbps.

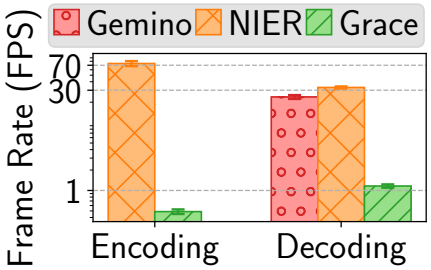


Figure 6.16: Frame rate of neural coding in Gemino [195], NIER, and Grace [36].

Decodable Frame Ratio under Packet losses. As shown in Figure 6.14, at 105 Kbps bandwidth, NIER's decodable frame ratio slightly drops from 99.96% to 99.73% when the loss rate increases from 0.05% to 10%, consistently outperforming the other schemes: at a 0.05% (10%) packet loss rate, NIER achieves 2.75x (530x), 1.06x (1.55x), and 2.18x (159x) average decodable frame ratio compared to *H264*, *VP8*, and *Gemino*, respectively. We have similar observations for the other bandwidth conditions.

Video Stall. Figure 6.15 reports the ratio of video stall time over the entire video duration under packet losses at 15 Kbps. As shown, under a 10% packet loss rate, NIER only incurs 0.34% video stall ratio (~0.2 seconds video stall for 1-minute streaming). *H264*, *VP8*, and *Gemino* suffer from significant video stall even under a 0.05% packet loss rate, making it impractical to apply them to real-time communication. We have similar observations for the other bandwidth conditions.

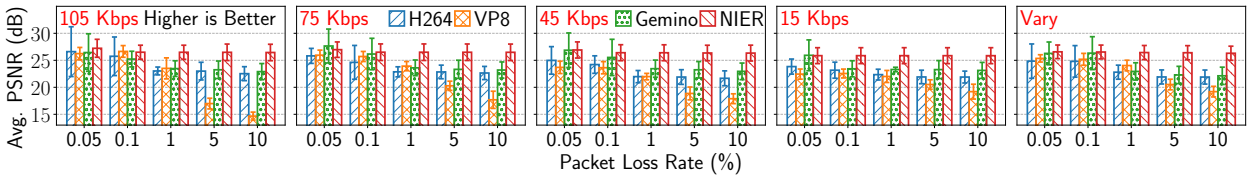


Figure 6.17: PSNR under packet losses, for stable (105, 75, 45, and 15 Kbps) and varying bandwidth (Vary).

Encoding/Decoding Frame Rate. We report frame rate of the neural coding components in *Gemino* (*i.e.,* decoding) and NIER. As shown in Figure 6.16, the *Gemino*'s computaionally heavy decoding phase achieves only ~24 FPS on our Macbook Air M1 2020 laptop, far below NIER's ~33 FPS.

Video Quality. Figure 6.17 show the PSNR of NIER and our comparison baselines under packet losses, for both stable and varying bandwidth. Recall that these 4 schemes bear hugely different decodable frame ratios under packet losses (see Figure 6.14), to fairly compare their visual quality, if a frame cannot be decoded, we use the most recent decoded frame as the current frame. As shown, for most of our investigated {bandwidth, packet loss rate} scenarios, NIER achieves the highest PSNR among all the schemes. In some cases where the packet loss rate is low (*e.g.,* 0.05%), *Gemino* has a comparative (or even better) visual quality compared to NIER. This can be attributed to the different neural-based approaches applied in NIER and *Gemino*: *Gemino* uses the entire low-resolution frame to generate its high-resolution frame, which provides much more information than the spare key-points used in NIER. This is also mentioned in [195]. We report the LPIPS [259] result in Figure 6.18, which draws a similar conclusion. Considering all the above dimensions, we believe NIER still provides the best overall QoE across all the scenarios.

Figure 6.18: LPIPS [259] under packet losses, for stable (105, 75, 45, and 15 Kbps) and varying bandwidth (Vary).

6.5.4 NIER vs. Loss-resilient Schemes

We compare NIER with **two loss-resilient video conferencing solutions**:

• ***Grace*** [36], a SOTA neural codec tailored for loss-resilient real-time video streaming. *Grace* jointly trains a neural encoder and decoder under a spectrum of simulated packet losses, allowing a graceful, less pronounced quality decrease when the packet loss rate increases. Specifically, the encoding of *Grace* consists of two stages, neural encoding and entropy coding. In the first stage, *Grace* applies its neural encoder to encode a frame to a tensor. The tensor is then split into non-overlapped sub-tensors, and *Grace* perform entropy coding on a per-sub-tenser basis. To decode a frame, *Grace* assembles the sub-tensors to a (corrupted) tensor and feeds it to its neural decoder. We first examine the **encoding/decoding frame rate** of *Grace*'s neural encoder and decoder. Since the authors open-source their code and pretrained models [213], we directly convert the most lightweight one (*64_freeze.model*) among them to an encoder and a decoder in the ONNX format [148], respectively. We evaluate their frame rate on our Macbook Air M1 2020. As shown in Figure 6.16, the encoding/decoding frame rate of *Grace*'s neural codec is far below the linerate (*i.e.,* 30+ FPS), achieving only ~0.5 FPS for encoding and ~1 FPS for decoding. We then access the **average bitrate** that *Grace* achieves in encoding our test videos by performing both *Grace*'s neural encoding and entropy coding [36] on them. Our result shows that the average bitrate across all our test videos after *Grace*'s encoding is 831.6 Kbps with a 48.5 Kbps

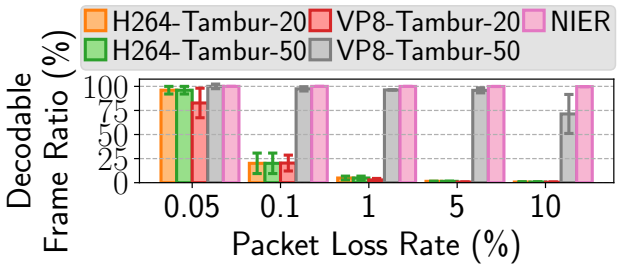


Figure 6.19: Decodable frame ratio of *Tambur* [180] and NIER under packet losses, at 105 Kbps.

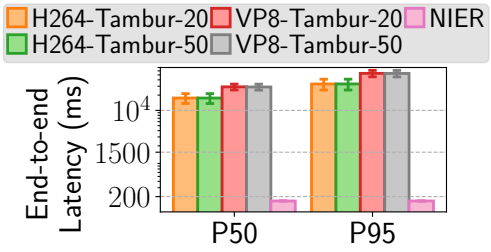


Figure 6.20: End-to-end latency of *Tambur* [180] and NIER at 105 Kbps.

standard deviation. The extremely low neural encoding/decoding frame rate and high encoding bitrate make it impractical to employ *Grace* in the context of low-bitrate video conferencing on commodity devices. We thus do not include any end-to-end performance comparison between *Grace* and NIER.

• ***Tambur*** [180], a SOTA FEC scheme based on streaming codes for video conferencing. *Tambur* also applies a machine learning model to adapt its redundancy rate based on the most recent packet loss measurement. Compared to traditional FEC schemes, *Tambur* improves the decodable frame ratio with the same amount of parity packets. We integrate the official Tambur library [180] into our NIER player (§6.4), and enable it on top of our *H264* and *VP8* baselines (§6.5.3). Since the authors do not open source their model and training dataset, we do not adaptively adjust *Tambur*'s redundancy ratio at runtime but use two fixed ratios: 20% and 50%. We thus involve 4 *Tambur* baselines, denoted as *H264-Tambur-20*, *H264-Tambur-50*, *VP8-Tambur-20*, and *VP8-Tambur-50*, respectively. We set a lower target bitrate to reserve bandwidth for FEC packets, *i.e.*,, for the redundancy ratio 20% (50%), we set the target bitrate to 0.8 (0.5) of the original value.

Decodable Frame Ratio. As shown in Figure 6.19, NIER consistently outperforms *Tambur* under all investigated packet losses (0.05, 0.1, 1, 5, and 10%), at 105 Kbps bandwidth. We notice

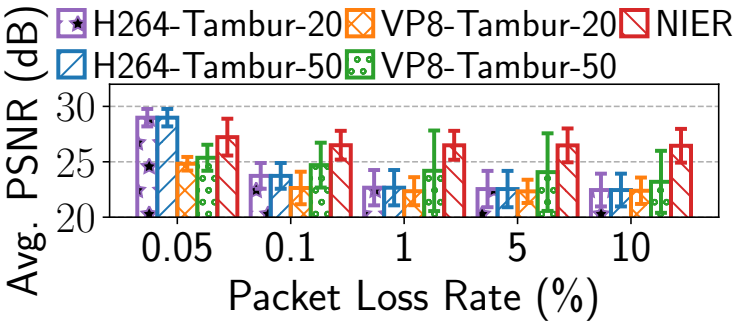


Figure 6.21: PSNR of *Tambur* [180] and NIER under packet losses, at 105 Kbps.

that in the low-bitrate scenario, *Tambur* does not appear as effective as we expect. We investigate the FEC packets provided by *Tambur* and find that *Tambur* has a minimum packet size of 311 Bytes, which in most cases is larger than the encoded size of a P-frame in *H264* and *VP8* in our setup. As a result, *Tambur-20* (*Tambur-50*) only outputs 1 (2) FEC packet(s) for a frame whose size is less than 311 Bytes, meaning that the protection is very limited.

End-to-end Latency. As shown in Figure 6.20, *Tambur* introduces substantial end-to-end latency to our *H264* and *VP8* baselines at 105 Kbps bandwidth (packet loss rate is set to 0), due to its large FEC packet size (as mentioned above).

Visual Quality. Figure 6.21 shows the PSNR of our *Tambur* baselines and NIER at 105 Kbps bandwidth. As shown, though *H264-Tambur-20*(-50) achieves better PSNR under low packet loss rate (*i.e.,* 0.05%), NIER outperforms *Tambur* in all the other scenarios. We thus believe NIER provides much better overal QoE for low-bitrate video conferencing compared to *Tambur*, considering all the above dimensions.

6.5.5 User Study

We conduct an IRB-approved user study to access real users' QoE when using NIER. We recruit 20 users with various demographics (8 males and 12 females; Ages vary between 20 and 30). We

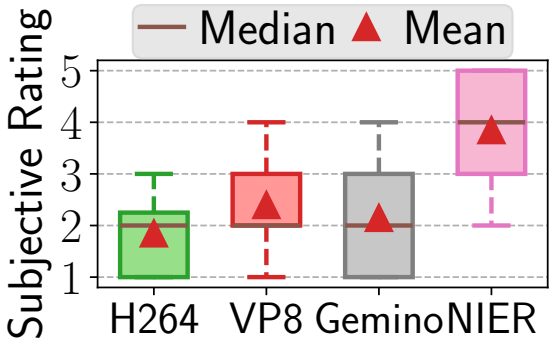


Figure 6.22: Subjective ratings of 20 users.

divide them into 10 groups, each consisting of 2 subjects. We let each group do a 1-minute 1-on-1 video call on our Macbook Air M1 2020 laptops (in 2 rooms), and subjectively rate their video call experience through 5 choices {1=very bad, 2=bad, 3=fair, 4=good, 5=very good}. Each group performs the above assessment four times. Each time we randomly apply one of our candidates: *H264*, *VP8*, *Gemino*, and NIER (details are in §6.5.3) for the video call, and we randomly pick a bandwidth in {75, 45} Kbps. Since in our user study we also transmit users' audio, we increase the bandwidth by 128 Kbps (the target encoding bitrate for our audio) to make sure there is sufficient bandwidth for audio delivery (the highest priority in our packet scheduling). We apply different packet loss rate, {0.05, 0.1, 1, 5, 10}%, to the 10 groups. As shown in Figure 6.22, compared to *H264*, *VP8*, and *Gemino*, NIER improves the subjective ratings by 2.0, 1.45, and 1.7, respectively. *VP8* outperforms *Gemino* in subjective rating probably due to *Gemino*'s substantial end-to-end latency.

6.5.6 Micro Benchmarks and Resource usage

We run experiments to show the effectiveness of NIER's design components, as well as NIER's resource usage.

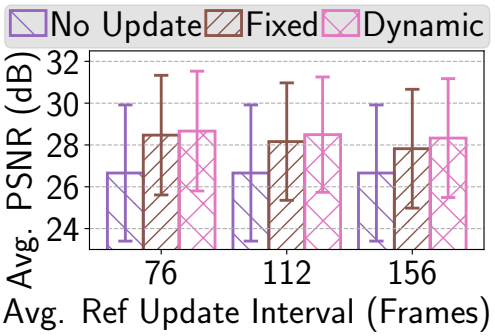


Figure 6.23: Dynamic vs. fixed reference update.

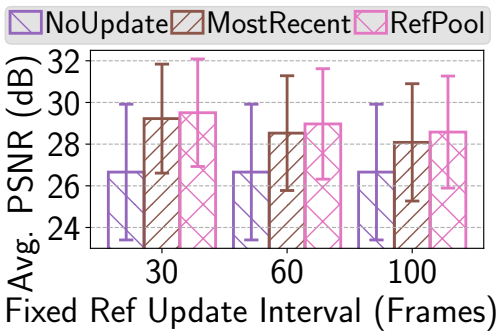


Figure 6.24: Reference pool vs. Using the most recent reference frame.

Reference Frame Update. We compare NIER's dynamic reference frame update with two baselines: (1) never updating the reference frame (*NoUpdate*), and (2) update the reference frame at a fixed pace (*Fixed*). For each video, we first apply *Dynamic* with 3 different minimum update intervals r (§6.3.1): 1, 2, and 3 seconds (*i.e.,* every 30, 60, 90 frames). We divide the number of reference frames over the total length of the video to get an average update interval (76, 112, and 156 as shown in Figure 6.23), which is applied to run *Fixed* on the same video. This makes sure *Dynamic* and *Fixed* consume the same bitrate to fairly compare them. Figure 6.23 shows that, compared to *NoUpdate*, *Fixed* improves the PSNR by 1.81, 1.50, and 1.16, for an average update interval (in frames) of 76, 112, and 156 frames. *Dynamic* further improves the PSNR by 0.19, 0.33, and 0.51 dB, for each interval, respectively, achieving 10.72%, 22.10%, and 43.64% improvement over *Fixed*.

Dynamic Reference Assignment. We compare NIER's dynamic reference assignment (*RefPool*) with two baselines: (1) never updating the reference frame (*NoUpdate*), and (2) always using the most recent reference frame for frame generation (*MostRecent*). We conduct the experiments with three fixed reference update intervals: *i.e.,* every 30, 60, and 100 frames. As shown in Figure 6.24, compared to *NoUpdate*, *MostRecent* improves the PSNR of the generated frames by 2.57,

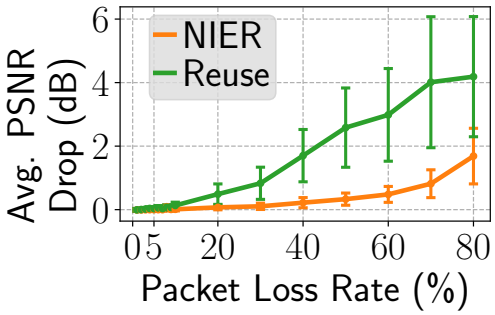


Figure 6.25: Missing key-point reconstruction (NIER) vs. most recent frame reuse (Reuse).

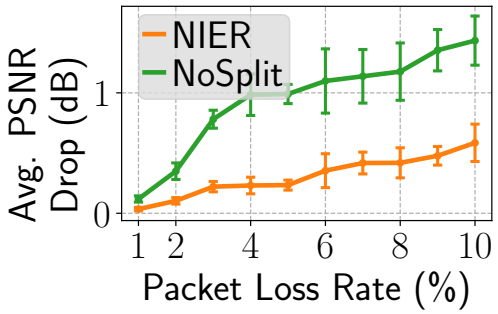


Figure 6.26: Reference split (NIER) vs. no split.

1.86, and 1.43 dB, when the updating interval is 30, 60, and 100, respectively. *RefPool* further improves PSNR by 0.28, 0.44, and 0.50 dB, for each interval, respectively, achieving 10%, 23.66%, and 34.97% improvement over *MostRecent*.

Missing Key-point Reconstruction. We compare NIER's missing key-point reconstruction with reusing the most recent frame (*Reuse*) under packet losses. Specifically, we randomly drop p% ($p \in [0, 80]$) key-point packets, and compare the quality drop of NIER and *Reuse* caused by packet losses. We use the first 300 frames for each of our test videos, and repeat the experiment 3 times for each p. As shown in Figure 6.25, compared to *Reuse*, NIER bear much less PSNR drop under packet losses, and can reduce the PSNR drop as much as 2.5 dB (a 59.7% reduction) under an 80% packet loss rate.

Reference Split. We compare NIER's reference frame split (NIER) with no split (*NoSplit*) under packet loss. Specifically, we randomly drop $p\%$ ($p \in [0, 10]$) reference frame packet (we use a fixed update interval of every 100 frames), and compare the quality drop of NIER and *NoSplit* caused by packet losses. As shown in Figure 6.26, compared to *NoSplit*, NIER bear much less drop under packet loss, reducing the PSNR drop by 0.85 dB (a 59.19% reduction), thanks to the resilience brought by reference split.

M_1	The vanilla key-point-based DIA framework
M_2	M_1 and removing redundant operations
M_3	M_2 and optimizing Key-point Extractor
M_4	M_3 and optimizing Motion Estimator
M_5	M_4 and optimizing Frame Generator
M_6	M_5 and using grayscale image (except frame generation)
M_7	M_6 and pipelining
M_8	M_7 and adding reconstruction loss for training

Table 6.3: Key-point-based DIA optimizations (cumulative).

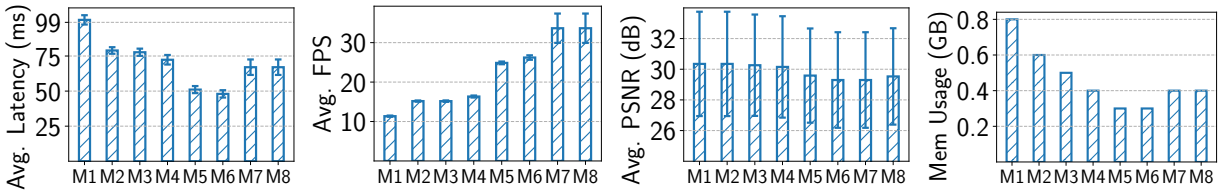


Figure 6.27: Average latency (per frame), frame rate (FPS), generation quality (PSNR), and peak memory usage of M_1 to M_8 in Table 6.3 (Apple MacBook Air M1 2020).

Effectiveness of Key-point-based DIA Acceleration. We examine the effectiveness of each of our proposed methods for accelerating key-point-based DIA. As listed in Table 6.3, M_1 denotes the vanilla DIA model (we use FOMM [193] to demonstrate) as the comparison baseline; M_2 to M_8 are proposed acceleration approaches in §6.3.5. They are presented in a *cumulative* fashion, *i.e.,* M_i includes every feature of M_{i-1} plus some new feature. Figure 6.27 shows the processing latency, frame rate, visual quality (in PSNR), and peak memory usage of FOMM on a Macbook Air M1 2020 laptop. As shown, compared to M_1, M_8 reduces the peak memory usage by 50%, reduces the processing latency by 34.7%, and improves the frame rate by 201%, with a slight PSNR drop (0.82 dB). Also, each optimization (M_2 to M_8) individually improves one or multiple metric(s).

6.6 Summary

Our current design bears several limitations. First, DIA requires training and may incur distortions in certain scenarios compared to traditional codecs. Note that this is a general issue for neural codecs, and the state-of-the-art is being steadily advanced by the CV/ML community. Second, NIER currently does not consider the scenario where multiple people appear in a frame. This can potentially be addressed by a more advanced DIA framework and/or additional system-level optimizations. Third, our current prototype and evaluations focus on 1-on-1 video calls. Scaling NIER to multi-party meetings involves more sophisticated design. Fourth, following the above points, whether NIER can be generalized from video conferencing, whose content mostly consists of human portraits, to general live streaming, requires more research.

Despite the above limitations, in this chapter, our work demonstrates the feasibility of building a practical neural-enhanced video conferencing system that simultaneously achieves low bitrate, reasonable visual quality, resilience to packet loss, and high frame rates on COTS devices. Our work suggests that familiar concepts in traditional multimedia systems (*e.g.,* ABR, multi-stream management, loss adaptation, layered coding) remain indispensable in the neural-enhanced regime, but *require a fresh revisit or even a completely new design.* Thanks to these efforts, our solution outperforms existing solutions in various categories (traditional video codecs, SR, FEC, loss-resilient neural codec, *etc.*) in one or more core metrics.

Chapter 7

Related Work

7.1 Volumetric Video Streaming and Super Resolution

Volumetric Video Streaming. There exist only a few studies on point-cloud-based volumetric video streaming [64, 68, 77, 102, 75, 63, 156, 165]. For example, DASH-PC [77] extends DASH to volumetric videos. PCC-DASH [75] is another DASH-based streaming scheme of compressed point clouds with bitrate adaptation support. ViVo [68] introduces visibility-aware optimizations for volumetric video streaming. GROOT [102] optimizes point cloud compression for volumetric videos. To the best of our knowledge, there is no existing work on applying 3D SR to volumetric video streaming.

Point Cloud SR. We classify existing work on point cloud SR into two categories: learning-based [106, 229, 234, 249] and optimization-based [8, 79]. Most learning-based approaches follow the workflow established in PU-Net [249], which divides a point cloud into patches, learns multi-level point features of each patch, expands the features, and reconstructs the points from the features. All the above methods are designed for a *single* point cloud; they suffer from numerous limitations when applied to volumetric videos (§2.1.1).

Visual Quality Assessment of Point Clouds. The state-of-the-art visual quality assessment focuses on *static, non-SR* point clouds [45, 136, 217]. For example, using a data-driven approach, Meynet *et al.* [136] present a full-reference visual quality metric for colored point clouds. Different from the above studies, we model the QoE of SR-enhanced volumetric video streaming. We address new challenges on modeling the impact of various factors such as the viewing distance, upsampling ratio, and SR incurred distortion (§3.3).

SR for Regular 2D Videos. NAS [246, 247] is one of the first proposals that apply 2D SR to Internet video streaming. Other recent efforts on 2D SR include PARSEC [42] for 360° panoramic video streaming, LiveNAS [91] for live video streaming, and NEMO [245] for mobile video streaming. In contrast, YuZu addresses numerous unique challenges (§3.1) on applying 3D SR to volumetric video streaming.

7.2 Immersive Content Delivery and mmWave

Immersive Content Delivery. We elaborate on some immersive content delivery systems mentioned in §4.5.4 (more can be found in [74]). Flare [166] and ViVo [68] apply viewport adaptation to optimize mobile 360° and volumetric videos streaming, respectively. M5 [258] investigates volumetric video streaming using adaptive mmWave beamforming. InstantReality [35] introduces a perceptual-aware approach to enhance VR media streaming. As a middleware framework, Habitus can be integrated into most of the above systems to enhance the application QoE (we have conducted a case study for ViVo in §4.7.7). Also note that Habitus is orthogonal to some VR systems (*e.g.*, MoVR [2]) that enhances the PHY layer (§4.1).

mmWave Throughput Prediction. Recent measurement studies have explored the feasibility of predicting mmWave throughput for various radios, such as commercial mmWave 5G [142], 802.11ad [5], and 802.11ay [237]. Lumos5G [142] establishes a composable machine learning framework to predict mmWave 5G throughput. Wu *et al.* uses Markov chain to predict the link quality of 802.11ay, based on the headset's motion data [237]. Aggarwal *et al.* conducts a measurement study on using a smartphone's motion sensor data to predict 802.11ad throughput [5]. They only consider 2-DoF (with a radio mounted on a guided rail) and LoS scenarios. None of the above studies employs the full-body pose, which we found to be an important feature for improving the prediction accuracy. Also, none of them conducts in-depth investigations on how to handle unseen changes as we do.

Multipath TCP Support for 802.11ac/ad. Despite a plethora of works on WiFi/cellular multipath [69, 235, 264], there are only a few studies on dual-band 802.11ac/ad multipath networking. MUST [201] predicts the best 60GHz beam and PHY rate setting, and switches between ac/ad links accordingly. We discussed MuSher [183], an MPTCP scheduler for 802.11ac/ad and compared it with Habitus in §4.7.4. Compared to the above works, Habitus is an application-layer solution designed specifically for immersive content delivery.

Improving mmWave Network Performance at PHY layer. There is rich literature on improving mmWave performance at the PHY layer, such as efficient beam selection [226], LiDAR-assisted beam management [233], and beam relay through smart metasurface [38], to name a few. Unlike the above, Habitus aims at optimizing the upper-layer network protocol stack for immersive content delivery without requiring modifying PHY-layer protocols.

7.3 Reducing Latency in Multimedia Streaming

Reducing Network-side Delay. There is extensive research on reducing network-side latency for various applications [133, 223, 204]. For example, Zhuge [133] reduces tail latency in wireless real-time communication (RTC) applications by proactively predicting network delay for each RTC packet and adjusting the packet sending rate accordingly. TwinStar [223] achieves low-latency video delivery by leveraging multiple network paths simultaneously to mitigate network jitter over a single path. Tan et al. [204] propose a data-driven LTE latency reduction framework for latency-sensitive mobile applications by analyzing operational LTE traces.

Adaptive Streaming. Adaptive bitrate (ABR) streaming is widely adopted in modern multimedia systems [197, 166, 68, 254], ranging from 2D videos to immersive content. The core concept is to encode multimedia content at multiple quality levels and stream the highest possible quality that matches the current network bandwidth [197, 254], ensuring timely delivery before the playback deadline. Some immersive video streaming systems [166, 68] adopt visibility-adaptive streaming, transmitting only content visible to the viewer. Machine-oriented systems, such as connected autonomous vehicles [261] and video analyst systems [92], dynamically adjust encoding parameters to adapt to fluctuating network conditions.

Optimizing Lossless Compression. A wide range of lossless compression techniques [6, 174, 196, 175, 30, 7] have been developed over the past decades. In addition, research has focused on accelerating existing frameworks [236, 191]. Wu et al. [236] propose an entropy-guided, content-aware pixel prioritization strategy to enhance the progressive compression performance of FLIF [196]. Shen et al. [191] optimize FLIF by leveraging GPU parallelism to accelerate compression. These studies are orthogonal to our work in Chapter 5.

7.4 Video Conferencing and Deep Image Animation

Low-bitrate Video Conferencing. There have been some studies on low-bitrate video conferencing. We elaborate Gemino [195] in §6.5.3. Wang *et al.* [224] achieves low-bitrate video calls by sending facial landmarks to the receiver, which applies affine transformation to the received facial landmarks along with a reference image to generate target frames. Agarwal *et al.* [4] and Oquab *et al.* [153] also achieve low bitrate by "encoding" video frames as key-points, using the deep image animation technique. However, they focus on improving the DIA models while overlooking the unique challenges in the networking and system dimensions.

Loss-resilient Real-time Streaming. Many recent studies [134, 180, 53, 107] have focused on loss-resilient real-time streaming. We elaborate Grace [36] and Tambur [180] in §6.5.4. Hairpin [134] optimizes retransmission and FEC strategies using a Markov decision process to minimize deadline misses and bandwidth costs. Salsify [53] integrates a video codec and transport protocol to adaptively mitigate packet loss and queueing delays. Reparo [107] leverages generative deep learning to reconstruct lost frames and enhance video quality. Despite these advancements, challenges such as computational cost, scalability, and bandwidth overhead limit their practical applicability in low-bitrate video conferencing.

Deep Image Animation. In recent years, researchers in the computer vision community have proposed various deep image animation models [192, 193, 194, 227, 117]. Monkey-Net [192] and FOMM [193] adopt deep neural models to animate a static image based on a reference image and sparse key-points extracted from a driving video. UVA [194] further improves the motion estimation, and uses motion disentanglement to enhance the quality of the generated images. Face-vid2vid [227] synthesizes the talking head based on the 2D one-shot reference. It extracted

the 3D key-point representation of a talking head to render novel-view images. Liu *et al.* [117] further incorporates the semantic information in image animation. Mallya *et al.* [127] proposes a cross-modal attention layer to identify correspondences between source and driving images, selecting and warping the most appropriate features.

Chapter 8

Conclusion and Future Work

My dissertation is dedicated to boosting the quality-of-experience (QoE) of emerging networked multimedia applications on existing network infrastructures through principled system designs, by addressing two major challenges: the evolution of existing network infrastructures worldwide lags behind the fast growth of multimedia traffic over the Internet, while each application exhibits unique characteristics in terms of data representation, resource requirements, and QoE assessment. In a word, the principled approach to enhancing QoE can be summarized as: developing adaptive, robust, and resource-efficient multimedia systems through innovative codecs, cross-layer optimizations, and user-centered QoE metrics, backed by solid prototyping and real-world evaluation. To explain, it includes:

• Exploring innovative coding schemes for multimedia content;

• Enhancing adaptability and robustness for streaming multimedia data under fluctuating network conditions;

• Conducting user studies to understand real users' QoE expectations for various applications;

• Leveraging cross-layer design;

• Seeking co-existence with commodity network infrastructures and minimizing the deployment effort.

I demonstrate how the principle can be applied and its effectiveness through in-depth studies of four representative networked multimedia applications: *volumetric video streaming*, *general mobile immersive content delivery*, *image live co-editing*, and *low-bitrate video conferencing*, where I design, implement, and thoroughly evaluate four innovative systems and frameworks, referred to as YuZu, Habitus, Alice, and NIER, each significantly improving users' QoE for its corresponding application. Specifically, to the best of my knowledge: YuZu is the first super-resolution-enhanced, QoE-aware volumetric video streaming system; Habitus is the first software framework aimming at optimizing the upper-layer network protocol stack for immersive content delivery (and metaverse applications in general); Alice is the first cross-platform compression adaptation framework for achieving low-latency image live co-editing under fluctuating network/computation resources; and NIER is the first practical low-bitrate video conferencing system enhanced by the key-point-based deep image animation technology and is suitable for a wide range of usage scenarios, in particular over challenging/metered networks.

I believe that my studies with the above applications, especially the principled approach, provide insights for a broader range of novel networked multimedia applications and can be easily adapted and applied to their system design and optimization.

8.1 Future Work

In the course of my research, I have noticed several limitations of my current work, such as the generalization concern for neural-based coding schemes, the potential issues when extending

some of the systems to multi-user use cases, and the requirement for accurate network condition estimation. In the near future, I am interested in further leveraging the aforementioned principle and diving deeper into networked multimedia system research, as elaborated in the following:

Toward Higher Generalizability. Several of my research works apply deep neural network models as the key building block to improve QoE for networked multimedia applications: YuZu leverages the super resolution (SR) technology to reduce the bandwidth requirement for streaming volumetric videos, Habitus uses Seq2Seq models to predict future mmWave throughput, and NIER adopts key-point-based deep image animation (DIA) models to achieve low-bitrate video conferencing. However, there are concerns about the generalizability (or performance drop) of DNN models at runtime, especially when processing the unseen data in the training dataset. For example, YuZu requires training SR models for each volumetric video, while NIER focuses on certain video conferencing scenarios (*i.e.,* a talking human with the upper body). In my future work, I plan to further enhance the generalizability of neural-based coding schemes, through three orthogonal directions: developing more advanced DNN models, leveraging transfer learning as we do when developing Habitus (Chapter 4), and exploring more sophisticated system designs.

Toward Higher Scalability. Though significantly boosting the QoE, some of my current works, such as Habitus (Chapter 4) and NIER (Chapter 6), focus on single-user or 1-on-1 video call use cases. Scaling these systems to multi-user use cases faces non-trivial challenges: when using Habitus, co-located users can impact the mmWave signals and thus the QoE of immersive content delivery among one another; and multi-party meeting may hugely increase the computation overhead for DIA models. In my future work, I plan to enhancing the scalability of these systems by addressing the above challenges through two orthogonal directions: developing more advanced DNN models, and involving additional system-level optimizations.

Toward Accurate Network Condition Estimation. To achieve adaptability, all networked multimedia systems rely on an accurate estimation of network condition (*e.g.,* bandwidth and packet loss rate). Nevertheless, throughout my research, I find that the current network condition estimation are still in their infancy, which severely degrades the QoE improvement brought by the optimizations in the other layers. In my future work, I plan to conduct research studies on network condition estimation over existing and future network infrastructures, with the ultimate goal of developing a unified and deployment-ready estimation framework that can benefit all networked multimedia systems.

Bibliography

[1] *8i Voxelized Full Bodies (8iVFB v2) - Dynamic Voxelized Point Cloud Dataset*. http://plenodb.jpeg.org/pc/8ilabs.

[2] Omid Abari, Dinesh Bharadia, Austin Duffield, and Dina Katabi. "Enabling {High-Quality} Untethered Virtual Reality". In: *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. 2017, pp. 531–544.

[3] *Adobe Photoshop*. https://www.adobe.com/products/photoshop.html.

[4] Madhav Agarwal, Anchit Gupta, Rudrabha Mukhopadhyay, Vinay P Namboodiri, and CV Jawahar. "Compressing video calls using synthetic talking heads". In: *arXiv preprint arXiv:2210.03692* (2022).

[5] Shivang Aggarwal, Zhaoning Kong, Moinak Ghoshal, Y Charlie Hu, and Dimitrios Koutsonikolas. "Throughput Prediction on 60 GHz Mobile Devices for High-Bandwidth, Latency-Sensitive Applications". In: *International Conference on Passive and Active Network Measurement*. Springer. 2021, pp. 513–528.

[6] Jyrki Alakuijala, Ruud Van Asseldonk, Sami Boukortt, Martin Bruse, Iulia-Maria Comșa, Moritz Firsching, Thomas Fischbacher, Evgenii Kliuchnikov, Sebastian Gomez, Robert Obryk, et al. "JPEG XL next-generation image compression architecture and coding tools". In: *Applications of digital image processing XLII*. Vol. 11137. SPIE. 2019, pp. 112–124.

[7] Umar Albalawi, Saraju P Mohanty, and Elias Kougianos. "A hardware architecture for better portable graphics (BPG) compression encoder". In: *2015 IEEE international Symposium on Nanoelectronic and information systems*. IEEE. 2015, pp. 291–296.

[8] Marc Alexa, Johannes Behr, Daniel Cohen-Or, Shachar Fleishman, David Levin, and Claudio T. Silva. "Computing and Rendering Point Set Surfaces". In: *IEEE Trans. on Visualization and Computer Graphics* 9.1 (2003), pp. 3–15.

[9] *Amazon Mechanical Turk*. https://www.mturk.com/.

[10] Apple Inc. *MacBook with M1 Chip*. https://www.apple.com/macbook-air-m1/. 2020.

[11] Bruno Artacho and Andreas Savakis. "Unipose: Unified human pose estimation in single images and videos". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 7035–7044.

[12] Armen S Asratian, Tristan MJ Denley, and Roland Häggkvist. *Bipartite graphs and their applications*. Vol. 131. Cambridge university press, 1998.

[13] Nicola Asuni and Andrea Giachetti. "TESTIMAGES: A large data archive for display and algorithm testing". In: *Journal of Graphics Tools* 17.4 (2013), pp. 113–125.

[14] Nicola Asuni and Andrea Giachetti. "TESTIMAGES: a Large-scale Archive for Testing Visual Devices and Basic Image Processing Algorithms." In: *STAG*. 2014, pp. 63–70.

[15] Ismail Avcibas, Bulent Sankur, and Khalid Sayood. "Statistical evaluation of image quality measures". In: (2001).

[16] *AWS Wavelength*. https://aws.amazon.com/wavelength/.

[17] Sławomir Bak, Etienne Corvee, Francois Bremond, and Monique Thonnat. "Person re-identification using spatial covariance regions of human body parts". In: *2010 7th IEEE International Conference on Advanced Video and Signal Based Surveillance*. IEEE. 2010, pp. 435–440.

[18] Athula Balachandran, Vyas Sekar, Aditya Akella, Srinivasan Seshan, Ion Stoica, and Hui Zhang. "Developing a Predictive Model of Quality of Experience for Internet Video". In: *Proceedings of ACM SIGCOMM*. 2013.

[19] Egon Balas and Paolo Toth. "Branch and bound methods for the traveling salesman problem". In: (1983).

[20] Jim Bankoski, Paul Wilkins, and Yaowu Xu. "Technical overview of VP8, an open source video codec for the web". In: *2011 IEEE International Conference on Multimedia and Expo*. IEEE. 2011, pp. 1–6.

[21] Valentin Bazarevsky and Ivan Grishchenko. "On-device, real-time body pose tracking with mediapipe blazepose". In: *Google AI Blog* (2020).

[22] *Bazel*. https://bazel.build/.

[23] Steven S. Beauchemin and John L. Barron. "The computation of optical flow". In: *ACM computing surveys (CSUR)* 27.3 (1995), pp. 433–466.

[24] Jacob Benesty, Jingdong Chen, Yiteng Huang, and Israel Cohen. "Pearson Correlation Coefficient". In: *Noise Reduction in Speech Processing*. 2009.

[25] Abdelhak Bentaleb, Bayan Taani, Ali C Begen, Christian Timmerer, and Roger Zimmermann. “A survey on bitrate adaptation schemes for streaming media over HTTP”. In: *IEEE Communications Surveys & Tutorials* 21.1 (2018), pp. 562–585.

[26] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. “Yolov4: Optimal speed and accuracy of object detection”. In: *arXiv preprint arXiv:2004.10934* (2020).

[27] Sergey Brin and Lawrence Page. “The anatomy of a large-scale hypertextual web search engine”. In: *Computer networks and ISDN systems* 30.1-7 (1998), pp. 107–117.

[28] Judith Butepage, Michael J Black, Danica Kragic, and Hedvig Kjellstrom. “Deep representation learning for human motion prediction and classification”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 6158–6166.

[29] Frank Cabello, Julio León, Yuzo Iano, and Rangel Arthur. “Implementation of a fixed-point 2D Gaussian Filter for Image Processing based on FPGA”. In: *2015 Signal Processing: Algorithms, Architectures, Arrangements, and Applications (SPA)*. IEEE. 2015, pp. 28–33.

[30] M Calore. *Meet WebP, Google’s New Image Format*. 2010.

[31] Zhe Cao, Tomas Simon, Shih-En Wei, and Yaser Sheikh. “Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields”. In: *CVPR*. 2017.

[32] Gaetano Carlucci, Luca De Cicco, Stefan Holmer, and Saverio Mascolo. “Analysis and design of the google congestion control for web real-time communication (WebRTC)”. In: *Proceedings of the 7th International Conference on Multimedia Systems*. 2016, pp. 1–12.

[33] Hong Chang, Dit-Yan Yeung, and Yimin Xiong. “Super-Resolution through Neighbor Embedding”. In: *Proceedings of CVPR*. 2004.

[34] Jiangong Chen, Xudong Qin, Guangyu Zhu, Bo Ji, and Bin Li. “Motion-prediction-based wireless scheduling for multi-user panoramic video streaming”. In: *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*. IEEE. 2021, pp. 1–10.

[35] Shaoyu Chen, Budmonde Duinkharjav, Xin Sun, Li-Yi Wei, Stefano Petrangeli, Jose Echevarria, Claudio Silva, and Qi Sun. “Instant reality: Gaze-contingent perceptual optimization for 3d virtual reality streaming”. In: *IEEE Transactions on Visualization and Computer Graphics* 28.5 (2022), pp. 2157–2167.

[36] Yihua Cheng, Ziyi Zhang, Hanchen Li, Anton Arapin, Yue Zhang, Qizheng Zhang, Yuhan Liu, Kuntai Du, Xu Zhang, Francis Y Yan, et al. “{GRACE}:{Loss-Resilient}{Real-Time} Video through Neural Codecs”. In: *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. 2024, pp. 509–531.

[37] Chung-Cheng Chiu, Tara N Sainath, Yonghui Wu, Rohit Prabhavalkar, Patrick Nguyen, Zhifeng Chen, Anjuli Kannan, Ron J Weiss, Kanishka Rao, Ekaterina Gonina, et al. "State-of-the-art speech recognition with sequence-to-sequence models". In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2018, pp. 4774–4778.

[38] Kun Woo Cho, Mohammad H Mazaheri, Jeremy Gummeson, Omid Abari, and Kyle Jamieson. "mmWall: A reconfigurable metamaterial surface for mmWave networks". In: *Proceedings of the 22nd International Workshop on Mobile Computing Systems and Applications*. 2021, pp. 119–125.

[39] David Roxbee Cox and Alan Stuart. "Some quick sign tests for trend in location and dispersion". In: *Biometrika* 42.1/2 (1955), pp. 80–95.

[40] *Create or Delete Presets*. https://helpx.adobe.com/photoshop-elements/prerelease/create-save-edit-delete-presets.html.

[41] Qiongjie Cui, Huaijiang Sun, Yue Kong, and Xiaoning Sun. "Deep Human Dynamics Prior". In: *Proceedings of the 29th ACM International Conference on Multimedia*. 2021, pp. 4371–4379.

[42] Mallesham Dasari, Arani Bhattacharya, Santiago Vargas, Pranjul Sahu, Aruna Balasubramanian, and Samir Das. "Streaming 360 degree Videos using Super-resolution". In: *Proceedings of IEEE INFOCOM*. 2020.

[43] *Demo Video for Dynamic Change Injection*. https://docs.google.com/presentation/d/e/2PACX-1vTokmujVFURIX6YBgyYz9aAjHUV9Ajr1a6dKqYHLmRbrcnI92flVAOO4TLOD338YXXUwyZOjIsSc0Hh/pub?start=false&loop=false&delayms=3000&slide=id.p.

[44] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. "Imagenet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[45] Rafael Diniz, Pedro Garcia Freitas, and Mylène C.Q. Farias. "Towards a Point Cloud Quality Assessment Model Using Local Binary Patterns". In: *Proceedings of the 12th International Conference on Quality of Multimedia Experience (QoMEX)*. 2020.

[46] *Draco 3D Data Compression*. https://google.github.io/draco/.

[47] *Facet: Photo Editor*. https://facet.ai/photo-editor. 2024.

[48] Nick Feamster and Hari Balakrishnan. "Packet loss recovery for streaming video". In: *12th International Packet Video Workshop*. PA: Pittsburgh. 2002, pp. 9–16.

[49] Wirth F Ferger. “The nature and use of the harmonic mean”. In: *Journal of the American Statistical Association* 26.173 (1931), pp. 36–40.

[50] *Figma: The Collaborative Interface Design Tool*. https://www.figma.com/. 2024.

[51] Alan Ford, Costin Raiciu, Mark J. Handley, Olivier Bonaventure, and Christoph Paasch. *TCP Extensions for Multipath Operation with Multiple Addresses*. RFC 8684. Mar. 2020. DOI: 10.17487/RFC8684.

[52] Steven Fortune. “Voronoi diagrams and Delaunay triangulations”. In: *Comp. in Euclidean Geometry*. World Sci., 1995, pp. 225–265.

[53] Sadjad Fouladi, John Emmons, Emre Orbay, Catherine Wu, Riad S Wahby, and Keith Winstein. “Salsify:{Low-Latency} network video through tighter integration between a video codec and a transport protocol”. In: *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. 2018, pp. 267–282.

[54] Xinyu Gao, Linglong Dai, Shuangfeng Han, I Chih-Lin, and Robert W Heath. “Energy-efficient hybrid analog and digital precoding for mmWave MIMO systems with large antenna arrays”. In: *IEEE Journal on Selected Areas in Communications* 34.4 (2016), pp. 998–1009.

[55] Yasaman Ghasempour, Claudio RCM Da Silva, Carlos Cordeiro, and Edward W Knightly. “IEEE 802.11 ay: Next-generation 60 GHz communication for 100 Gb/s Wi-Fi”. In: *IEEE Communications Magazine* 55.12 (2017), pp. 186–192.

[56] *Git Repositories on WebRTC*. URL: https://webrtc.googlesource.com/.

[57] Enrico Gobbetti and Fabio Marton. “Far voxels: a multiresolution framework for interactive rendering of huge complex 3d models on commodity graphics platforms”. In: *ACM SIGGRAPH 2005 Papers*. 2005, pp. 878–885.

[58] *Google ARCore*. https://developers.google.com/ar.

[59] *Google ARCore Motion Tracking*. https://developers.google.com/ar/develop/fundamentals.

[60] *Google Docs: Online Document Editor*. https://www.google.com/docs/about/. 2024.

[61] Yongjie Guan, Xueyu Hou, Nan Wu, Bo Han, and Tao Han. “DeepMix: mobility-aware, lightweight, and hybrid 3D object detection for headsets”. In: *Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services*. 2022, pp. 28–41.

[62] Yu Guan, Chengyuan Zheng, Xinggong Zhang, Zongming Guo, and Junchen Jiang. "Pano: Optimizing 360 video streaming with a better understanding of quality perception". In: *Proceedings of the ACM Special Interest Group on Data Communication*. 2019, pp. 394–407.

[63] Serhan Gül, Dimitri Podborski, Thomas Buchholz, Thomas Schierl, and Cornelius Hellge. "Low-latency cloud-based volumetric video streaming using head motion prediction". In: *Proceedings of the 30th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*. 2020, pp. 27–33.

[64] Serhan Gül, Dimitri Podborski, Jangwoo Son, Gurdeep Singh Bhullar, Thomas Buchholz, Thomas Schierl, and Cornelius Hellge. "Cloud Rendering-based Volumetric Video Streaming System for Mixed Reality Services". In: *Proceedings of ACM MMSys*. 2020.

[65] Andreas Haas, Andreas Rossberg, Derek L Schuff, Ben L Titzer, Michael Holman, Dan Gohman, Luke Wagner, Alon Zakai, and JF Bastien. "Bringing the web up to speed with WebAssembly". In: *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*. 2017, pp. 185–200.

[66] *Habitus Code*. https://github.com/zhan6841/Habitus-open-sourced-code.git.

[67] *Habitus Data*. https://drive.google.com/drive/folders/1B7rlIQG6ycg2OFML3CHxoPNge56xPhq4.

[68] Bo Han, Yu Liu, and Feng Qian. "ViVo: Visibility-Aware Mobile Volumetric Video Streaming". In: *Proceedings of ACM MobiCom*. 2020.

[69] Bo Han, Feng Qian, Lusheng Ji, and Vijay Gopalakrishnan. "MP-DASH: Adaptive video streaming over preference-aware multipath". In: *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*. 2016, pp. 129–143.

[70] Ahmad Hassan, Arvind Narayanan, Anlan Zhang, Wei Ye, Ruiyang Zhu, Shuowei Jin, Jason Carpenter, Z Morley Mao, Feng Qian, and Zhi-Li Zhang. "Vivisecting mobility management in 5G cellular networks". In: *Proceedings of the ACM SIGCOMM 2022 Conference*. 2022, pp. 86–100.

[71] Jian He, Mubashir Adnan Qureshi, Lili Qiu, Jin Li, Feng Li, and Lei Han. "Rubiks: Practical 360° Streaming for Smartphones". In: *Proceedings of ACM MobiSys*. 2018.

[72] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[73] Qi He, Constantine Dovrolis, and Mostafa Ammar. "On the predictability of large transfer TCP throughput". In: *ACM SIGCOMM Computer Communication Review* 35.4 (2005), pp. 145–156.

[74] Jeroen van der Hooft, Hadi Amirpour, Maria Torres Vega, Yago Sanchez, Raimund Schatz, Thomas Schierl, and Christian Timmerer. “A Tutorial on Immersive Video Delivery: From Omnidirectional Video to Holography”. In: *IEEE Communications Surveys & Tutorials* (2023).

[75] Jeroen van der Hooft, Tim Wauters, Filip De Turck, Christian Timmerer, and Hermann Hellwagner. “Towards 6DoF HTTP Adaptive Streaming Through Point Cloud Compression”. In: *Proceedings of ACM Multimedia*. 2019.

[76] Alain Hore and Djemel Ziou. “Image quality metrics: PSNR vs. SSIM”. In: *Proceedings of the 20th International Conference on Pattern Recognition*. IEEE. 2010, pp. 2366–2369.

[77] Mohammad Hosseini and Christian Timmerer. “Dynamic Adaptive Point Cloud Streaming”. In: *Proceedings of ACM Packet Video*. 2018.

[78] Bin Hu, Xumiao Zhang, Qixin Zhang, Nitin Varyani, Z Morley Mao, Feng Qian, and Zhi-Li Zhang. “LEO Satellite vs. Cellular Networks: Exploring the Potential for Synergistic Integration”. In: *Companion of the 19th International Conference on emerging Networking EXperiments and Technologies*. 2023, pp. 45–51.

[79] Hui Huang, Shihao Wu, Minglun Gong, Daniel Cohen-Or, Uri Ascher, and Hao Zhang. “Edge-Aware Point Set Resampling”. In: *ACM Transactions on Graphics* 32.1 (2013), 9:1–9:12.

[80] Yan Huang, Jingliang Peng, C.-C. Jay Kuo, and M. Gopi. “A Generic Scheme for Progressive Point Cloud Coding”. In: *IEEE Trans. on Vis. and Computer Graphics* 14.2 (2008), pp. 440–453.

[81] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. “A Buffer-Based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service”. In: *Proceedings of ACM SIGCOMM*. 2014.

[82] Erik Hubo, Tom Mertens, Tom Haber, and Philippe Bekaert. “The Quantized kd-Tree: Efficient Ray Tracing of Compressed Point Clouds”. In: *Proceedings of IEEE Symposium on Interactive Ray Tracing*. 2006.

[83] C Huitema. *Real time control protocol (RTCP) attribute in session description protocol (SDP)*. Tech. rep. 2003.

[84] *IEEE MMSP 2020 Challenge: Learning-based Image Coding Challenge Test Images*. https://jpegai.github.io/test_images/.

[85] *ITU-P.913: Methods for the subjective assessment of video quality, audio quality and audiovisual quality of Internet video and distribution quality television in any environment*. https://www.itu.int/rec/T-REC-P.913.

[86] Junchen Jiang, Vyas Sekar, and Hui Zhang. “Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive”. In: *Proceedings of the 8th international conference on Emerging networking experiments and technologies*. 2012, pp. 97–108.

[87] Hanbyul Joo, Tomas Simon, Xulong Li, Hao Liu, Lei Tan, Lin Gui, Sean Banerjee, Timothy Scott Godisart, Bart Nabbe, Iain Matthews, Takeo Kanade, Shohei Nobuhara, and Yaser Sheikh. “Panoptic Studio: A Massively Multiview System for Social Interaction Capture”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2017).

[88] Julius Kammerl, Nico Blodow, Radu Bogdan Rusu, Suat Gedikli, Michael Beetz, and Eckehard Steinbach. “Real-time Compression of Point Cloud Streams”. In: *Proceedings of International Conference on Robotics and Automation*. 2012.

[89] *KCXGHYI VR Headset*. https://bit.ly/41UMh19.

[90] Ali Safari Khatouni, Marco Mellia, Marco Ajmone Marsan, Stefan Alfredsson, Jonas Karlsson, Anna Brunstrom, Ozgu Alay, Andra Lutu, Cise Midoglu, and Vincenzo Mancuso. “Speedtest-like measurements in 3g/4g networks: The monroe experience”. In: *Proceedings of the 29th International Teletraffic Congress (ITC 29)*. 2017, pp. 169–177.

[91] Jaehong Kim, Youngmok Jung, Hyunho Yeo, Juncheol Ye, and Dongsu Han. “Neural-Enhanced Live Streaming: Improving Live Video Ingest via Online Learning”. In: *Proceedings of ACM SIGCOMM*. 2020.

[92] Seyeon Kim, Kyungmin Bin, Donggyu Yang, Sangtae Ha, Song Chong, and Kyunghan Lee. “ENTRO: Tackling the Encoding and Networking Trade-off in Offloaded Video Analytics”. In: *Proceedings of the 31st ACM International Conference on Multimedia*. 2023, pp. 9115–9123.

[93] Earl J Kirkland and Earl J Kirkland. “Bilinear interpolation”. In: *Advanced computing in electron microscopy* (2010), pp. 261–263.

[94] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi. “Optimization by Simulated Annealing”. In: *Science* 220.4598 (1983), pp. 671–680.

[95] *Kodak Lossless True Color Image Suite*. https://r0k.us/graphics/kodak/.

[96] Goluck Konuko, Giuseppe Valenzise, and Stéphane Lathuilière. “Ultra-low bitrate video conferencing using deep image animation”. In: *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2021, pp. 4210–4214.

[97] Wouter M Kouw and Marco Loog. “An introduction to domain adaptation and transfer learning”. In: *arXiv preprint arXiv:1812.11806* (2018).

[98] Max Ku, Tianle Li, Kai Zhang, Yujie Lu, Xingyu Fu, Wenwen Zhuang, and Wenhu Chen. "Imagenhub: Standardizing the evaluation of conditional image generation models". In: *arXiv preprint arXiv:2310.01596* (2023).

[99] Taras Kucherenko, Jonas Beskow, and Hedvig Kjellström. "A neural network approach to missing marker reconstruction in human motion capture". In: *arXiv preprint arXiv:1803.02665* (2018).

[100] Zeqi Lai, Y Charlie Hu, Yong Cui, Linhui Sun, Ningwei Dai, and Hung-Sheng Lee. "Furion: Engineering high-quality immersive virtual reality on today's mobile devices". In: *IEEE Transactions on Mobile Computing* 19.7 (2019), pp. 1586–1602.

[101] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, et al. "The quic transport protocol: Design and internet-scale deployment". In: *Proceedings of the conference of the ACM special interest group on data communication*. 2017, pp. 183–196.

[102] Kyungjin Lee, Juheon Yi, Youngki Lee, Sunghyun Choi, and Young Min Kim. "GROOT: A Real-Time Streaming System of High-Fidelity Volumetric Videos". In: *Proceedings of ACM MobiCom*. 2020.

[103] Lik-Hang Lee, Tristan Braud, Pengyuan Zhou, Lin Wang, Dianlei Xu, Zijun Lin, Abhishek Kumar, Carlos Bermejo, and Pan Hui. "All one needs to know about metaverse: A complete survey on technological singularity, virtual ecosystem, and research agenda". In: *arXiv preprint arXiv:2110.05352* (2021).

[104] Pai-Feng Lee, Chi-Kang Kao, Juin-Ling Tseng, Bin-Shyan Jong, and Tsong-Wuu Lin. "3D animation compression using affine transformation matrix and principal component analysis". In: *IEICE TRANSACTIONS on Information and Systems* 90.7 (2007), pp. 1073–1084.

[105] Huan Lei, Naveed Akhtar, and Ajmal Mian. "Spherical Kernel for Efficient Graph Convolution on 3D Point Clouds". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2020).

[106] Ruihui Li, Xianzhi Li, Chi-Wing Fu, Daniel Cohen-Or, and Pheng-Ann Heng. "PU-GAN: A Point Cloud Upsampling Adversarial Network". In: *Proceedings of ICCV*. 2019.

[107] Tianhong Li, Vibhaalakshmi Sivaraman, Pantea Karimi, Lijie Fan, Mohammad Alizadeh, and Dina Katabi. "Reparo: Loss-Resilient Generative Codec for Video Conferencing". In: *arXiv preprint arXiv:2305.14135* (2023).

[108] Yang Li, Hao Lin, Zhenhua Li, Yunhao Liu, Feng Qian, Liangyi Gong, Xianlong Xin, and Tianyin Xu. "A nationwide study on cellular reliability: Measurement, analysis, and enhancements". In: *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*. 2021, pp. 597–609.

[109] *libdatachannel*. URL: https://libdatachannel.org/.

[110] *libjxl v0.10.0*. https://github.com/libjxl/libjxl/archive/refs/tags/v0.10.0.zip.

[111] *Libpcap*. https://www.tcpdump.org.

[112] *libpng v1.6.43*. https://github.com/pnggroup/libpng/releases/tag/v1.6.43.

[113] *libsoundio*. URL: http://libsound.io/.

[114] Jyh-Ming Lien, Gregorij Kurillo, and Ruzena Bajcsy. "Multi-camera tele-immersion system with real-time model driven data compression". In: *The Visual Computer* 26.3 (2010), pp. 3–15.

[115] *Linux TC Man Page*. https://linux.die.net/man/8/tc.

[116] *Linux Wireless*. https://wireless.wiki.kernel.org/en/users/documentation/iw.

[117] Kangning Liu, Yu-Chuan Su, Ruijin Cang, Xuhui Jia, et al. "Controllable One-Shot Face Video Synthesis With Semantic Aware Prior". In: *arXiv preprint arXiv:2304.14471* (2023).

[118] Xing Liu, Christina Vlachou, Feng Qian, Chendong Wang, and Kyu-Han Kim. "Firefly: Untethered multi-user VR for commodity mobile devices". In: *Proceedings of the 2020 USENIX Conference on Usenix Annual Technical Conference*. 2020, pp. 943–657.

[119] Yu Liu, Bo Han, Feng Qian, Arvind Narayanan, and Zhi-Li Zhang. "Vues: Practical Mobile Volumetric Video Streaming Through Multiview Transcoding". In: *ACM MobiCom 2022* (2022).

[120] Salvatore Loreto and Simon Pietro Romano. *Real-time communication with WebRTC: peer-to-peer in the browser*. " O'Reilly Media, Inc.", 2014.

[121] *Lossless Compression Software for Camera Raw Images*. https://imagecompression.info/.

[122] Louca. "Implementation of IEEE single precision floating point addition and multiplication on FPGAs". In: *1996 Proceedings IEEE Symposium on FPGAs for Custom Computing Machines*. IEEE. 1996, pp. 107–116.

[123] Yiqing Ma, Han Tian, Xudong Liao, Junxue Zhang, Weiyan Wang, Kai Chen, and Xin Jin. "Multi-objective congestion control". In: *Proceedings of the Seventeenth European Conference on Computer Systems*. 2022, pp. 218–235.

[124] *MacBook Air (M1, 2020) - Technical Specifications*. https://support.apple.com/en-us/111883.

[125] Kyle MacMillan, Tarun Mangla, James Saxon, and Nick Feamster. “Measuring the performance and network utilization of popular video conferencing applications”. In: *Proceedings of the 21st ACM Internet Measurement Conference*. 2021, pp. 229–244.

[126] *Magic Leap One*. https://www.magicleap.com/en-us/magic-leap-1.

[127] Arun Mallya, Ting-Chun Wang, and Ming-Yu Liu. “Implicit warping for animation with image sets”. In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 22438–22450.

[128] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. “Neural Adaptive Video Streaming with Pensieve”. In: *Proceedings of ACM SIGCOMM*. 2017.

[129] Julieta Martinez, Michael J Black, and Javier Romero. “On human motion prediction using recurrent neural networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 2891–2900.

[130] Steven McCanne and Van Jacobson. “The BSD Packet Filter: A New Architecture for User-level Packet Capture.” In: *USENIX winter*. Vol. 46. Citeseer. 1993, pp. 259–270.

[131] *Measuring Broadband America Mobile Data*. https://www.fcc.gov/reports-research/reports/measuring-broadband-america/measuring-broadband-america-mobile-data. 2023.

[132] Rufael Mekuria, Kees Blom, and Pablo Cesar. “Design, Implementation and Evaluation of a Point Cloud Codec for Tele-Immersive Video”. In: *IEEE Trans. on Circuits and Systems for Video Technology* 27.4 (2017), pp. 828–842.

[133] Zili Meng, Yaning Guo, Chen Sun, Bo Wang, Justine Sherry, Hongqiang Harry Liu, and Mingwei Xu. “Achieving consistent low latency for wireless real-time communications with the shortest control loop”. In: *Proceedings of the ACM SIGCOMM 2022 Conference*. 2022, pp. 193–206.

[134] Zili Meng, Xiao Kong, Jing Chen, Bo Wang, Mingwei Xu, Rui Han, Honghao Liu, Venkat Arun, Hongxin Hu, and Xue Wei. “Hairpin: Rethinking packet loss recovery in edge-based interactive video streaming”. In: *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. 2024, pp. 907–926.

[135] *Metaverse*. https://about.facebook.com/metaverse/.

[136] Gabriel Meynet, Yana Nehmé, Julie Digne, and Guillaume Lavoué. “PCQM: A Full-Reference Quality Metric for Colored 3D Point Clouds”. In: *Proceedings of the 12th International Conference on Quality of Multimedia Experience (QoMEX)*. 2020.

[137] Carsten Moenning and Neil A Dodgson. *Fast marching farthest point sampling*. Tech. rep. University of Cambridge, 2003.

[138] *MPU*. https://github.com/yifita/3PU.

[139] Soraia R Musse, Christian Babski, Tolga Capin, and Daniel Thalmann. "Crowd modelling in collaborative virtual environments". In: *Proceedings of the ACM symposium on Virtual reality software and technology*. 1998, pp. 115–123.

[140] Stylianos Mystakidis. "Metaverse". In: *Encyclopedia* 2.1 (2022), pp. 486–497.

[141] Abdelhamid Nafaa, Tarik Taleb, and Liam Murphy. "Forward error correction strategies for media streaming over wireless networks". In: *IEEE Communications Magazine* 46.1 (2008), pp. 72–79.

[142] Arvind Narayanan, Eman Ramadan, Rishabh Mehta, Xinyue Hu, Qingxu Liu, Rostand AK Fezeu, Udhaya Kumar Dayalan, Saurabh Verma, Peiqi Ji, Tao Li, et al. "Lumos5G: Mapping and predicting commercial mmWave 5G throughput". In: *Proceedings of the ACM Internet Measurement Conference*. 2020, pp. 176–193.

[143] Jiri Navratil and R Les Cottrell. "ABwE: A practical approach to available bandwidth estimation". In: *Proceedings of the 4th Passive and Active Measurement Workshop PAM 2003*. Citeseer. 2003.

[144] *Netgear Nighthawk X10*. https://www.netgear.com/support/product/r9000.aspx.

[145] Ravi Netravali, Anirudh Sivaraman, Somak Das, Ameesh Goyal, Keith Winstein, James Mickens, and Hari Balakrishnan. "Mahimahi: accurate {Record-and-Replay} for {HTTP}". In: *2015 USENIX Annual Technical Conference (USENIX ATC 15)*. 2015, pp. 417–429.

[146] Gerd Niestegge. "The 'leaky bucket' policing method in the atm (asynchronous transfer mode) network". In: *International Journal of Digital & Analog Communication Systems* 3.2 (1990), pp. 187–197.

[147] Thomas Nitsche, Carlos Cordeiro, Adriana B Flores, Edward W Knightly, Eldad Perahia, and Joerg C Widmer. "IEEE 802.11 ad: directional 60 GHz communication for multi-Gigabit-per-second Wi-Fi". In: *IEEE Communications Magazine* 52.12 (2014), pp. 132–141.

[148] ONNX Community. *ONNX | Home*. https://onnx.ai/. 2019.

[149] *OpenCV*. URL: https://opencv.org/.

[150] *OpenCV 4.9.0*. https://github.com/opencv/opencv/releases/tag/4.9.0.

[151] *OpenPose Codebase*. https://github.com/CMU-Perceptual-Computing-Lab/openpose.

[152] *Opus*. URL: https://github.com/xiph/opus.

[153] Maxime Oquab, Pierre Stock, Daniel Haziza, Tao Xu, Peizhao Zhang, Onur Celebi, Yana Hasson, Patrick Labatut, Bobo Bose-Kolanu, Thibault Peyronel, et al. “Low bandwidth video-chat compression using deep generative models”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 2388–2397.

[154] Sergio Orts-Escolano, Christoph Rhemann, Sean Fanello, Wayne Chang, Adarsh Kowdle, Yury Degtyarev, David Kim, Philip Davidson, Sameh Khamis, Mingsong Dou, Vladimir Tankovich, Charles Loop, Qin Cai, Philip Chou, Sarah Mennicken, Julien Valentin, Vivek Pradeep, Shenlong Wang, Sing Bing Kang, Pushmeet Kohli, Yuliya Lutchyn, Cem Keskin, and Shahram Izadi. “Holoportation: Virtual 3D Teleportation in Real-time”. In: *Proceedings of ACM UIST*. 2016.

[155] *Overleaf, Online LaTeX Editor*. https://www.overleaf.com/. 2024.

[156] Jounsup Park, Philip A Chou, and Jenq-Neng Hwang. “Volumetric media streaming for augmented reality”. In: *2018 IEEE Global communications conference (GLOBECOM)*. IEEE. 2018, pp. 1–6.

[157] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. “Automatic differentiation in PyTorch”. In: (2017).

[158] Stefano Petrangeli, Dries Pauwels, Jeroen van der Hooft, Tim Wauters, Filip De Turck, and Jürgen Slowack. “Improving quality and scalability of WebRTC video collaboration applications”. In: *Proceedings of the 9th ACM Multimedia Systems Conference*. 2018, pp. 533–536.

[159] *Photoshop Unlocks Creative Collaboration with Live Co-Editing*. https://blog.adobe.com/en/publish/2025/01/14/photoshop-unlocks-creative-collaboration-with-live-co-editing-join-private-beta.

[160] Dustin Podell, Zion English, Kyle Lacey, Andreas Blattmann, Tim Dockhorn, Jonas Müller, Joe Penna, and Robin Rombach. “Sdxl: Improving latent diffusion models for high-resolution image synthesis”. In: *arXiv preprint arXiv:2307.01952* (2023).

[161] *Protocol Buffers*. https://developers.google.com/protocol-buffers.

[162] *PU-GAN*. https://github.com/liruihui/PU-GAN.

[163] *Pytorch*. https://pytorch.org.

[164] Charles R. Qi, Li Yi, Hao Su, and Leonidas J. Guibas. “PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space”. In: *Proceedings of Conference on Neural Information Processing Systems (NeurIPS)*. 2017.

[165] Feng Qian, Bo Han, Jarrell Pair, and Vijay Gopalakrishnan. “Toward Practical Volumetric Video Streaming On Commodity Smartphones”. In: *Proceedings of ACM HotMobile*. 2019.

[166] Feng Qian, Bo Han, Qingyang Xiao, and Vijay Gopalakrishnan. “Flare: Practical Viewport-Adaptive 360-Degree Video Streaming for Mobile Devices”. In: *Proceedings of ACM MobiCom*. 2018.

[167] *Qualtrics experience management platform*. https://www.qualtrics.com/.

[168] Vasanthan Raghavan, Andrzej Partyka, Lida Akhoondzadeh-Asl, Mohammad Ali Tassoudji, Ozge Hizir Koymen, and John Sanelli. “Millimeter wave channel measurements and implications for PHY layer design”. In: *IEEE Transactions on Antennas and Propagation* 65.12 (2017), pp. 6521–6533.

[169] Eman Ramadan, Arvind Narayanan, Udhaya Kumar Dayalan, Rostand AK Fezeu, Feng Qian, and Zhi-Li Zhang. “Case for 5G-aware video streaming applications”. In: *Proceedings of the 1st workshop on 5g measurements, modeling, and use cases*. 2021, pp. 27–34.

[170] Davis Rempe, Tolga Birdal, Aaron Hertzmann, Jimei Yang, Srinath Sridhar, and Leonidas J Guibas. “Humor: 3d human motion model for robust pose estimation”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 11488–11499.

[171] Vinay Joseph Ribeiro, Rudolf H Riedi, Richard G Baraniuk, Jiri Navratil, and Les Cottrell. “pathchirp: Efficient available bandwidth estimation for network paths”. In: *Passive and active measurement workshop*. 2003.

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

[173] Luigi Rizzo. “Dummynet: a simple approach to the evaluation of network protocols”. In: *ACM SIGCOMM Computer Communication Review* 27.1 (1997), pp. 31–41.

[174] Greg Roelofs. *PNG: the definitive guide*. O’Reilly & Associates, Inc., 1999.

[175] Greg Roelofs. “zlib: A massively spiffy yet delicately unobtrusive compression library”. In: *http://www. zlib. net/* (2017).

[176] *ROG Phone II*. https://rog.asus.com/phones/rog-phone-ii-model.

[177] *Rokoko Smartsuit Pro II*. https://www.rokoko.com/products/smartsuit-pro.

[178] Jonathan Rosenberg. *RFC 5245: Interactive connectivity establishment (ICE): A protocol for network address translator (NAT) traversal for offer/answer protocols*. 2010.

[179] Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. “The earth mover’s distance as a metric for image retrieval”. In: *International journal of computer vision* 40.2 (2000), pp. 99–121.

[180] Michael Rudow, Francis Y Yan, Abhishek Kumar, Ganesh Ananthanarayanan, Martin Ellis, and KV Rashmi. “Tambur: Efficient loss recovery for videoconferencing via streaming codes”. In: *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 2023, pp. 953–971.

[181] Agnieszka Rutkowska. “Properties of the Cox–Stuart test for trend in application to hydrological series: the simulation study”. In: *Communications in Statistics-Simulation and Computation* 44.3 (2015), pp. 565–579.

[182] Andreas Sackl, Pedro Casas, Raimund Schatz, Lucjan Janowski, and Ralf Irmer. “Quantifying the impact of network bandwidth fluctuations and outages on web qoe”. In: *2015 Seventh international workshop on quality of multimedia experience (QoMEX)*. IEEE. 2015, pp. 1–6.

[183] Swetank Kumar Saha, Shivang Aggarwal, Rohan Pathak, Dimitrios Koutsonikolas, and Joerg Widmer. “Musher: An agile multipath-TCP scheduler for dual-band 802.11 ad/ac wireless LANs”. In: *The 25th Annual International Conference on Mobile Computing and Networking*. 2019, pp. 1–16.

[184] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. “Mobilenetv2: Inverted residuals and linear bottlenecks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 4510–4520.

[185] Katsuyoshi Sato, Takeshi Manabe, Toshio Ihara, Hiroshi Saito, Shigeru Ito, Tetsu Tanaka, Kazuyoshi Sugai, Norichika Ohmi, Yasushi Murakami, Masanori Shibayama, et al. “Measurements of reflection and transmission characteristics of interior structures of office building in the 60-GHz band”. In: *IEEE transactions on antennas and propagation* 45.12 (1997), pp. 1783–1792.

[186] Heiko Schwarz, Detlev Marpe, and Thomas Wiegand. “Overview of the scalable video coding extension of the H. 264/AVC standard”. In: *IEEE Transactions on circuits and systems for video technology* 17.9 (2007), pp. 1103–1120.

[187] Paul Scovanner, Saad Ali, and Mubarak Shah. “A 3-dimensional SIFT descriptor and its application to action recognition”. In: *Proceedings of the 15th ACM International Conference on Multimedia*. 2007, pp. 357–360.

[188] SDL. *Simple DirectMedia Layer 3.0*. https://wiki.libsdl.org/SDL3/FrontPage. 2024.

[189] Claude Elwood Shannon. “A mathematical theory of communication”. In: *The Bell system technical journal* 27.3 (1948), pp. 379–423.

[190] Yiru Shen, Chen Feng, Yaoqing Yang, and Dong Tian. “Mining Point Cloud Local Structures by Kernel Correlation and Graph Pooling”. In: *Proceedings of CVPR*. 2018.

[191] Yu Shen, Gang Wu, Vishy Swaminathan, Haoliang Wang, Stefano Petrangeli, and Tong Yu. “GPU-accelerated Lossless Image Compression with Massive Parallelization”. In: *2023 IEEE International Symposium on Multimedia (ISM)*. IEEE. 2023, pp. 321–324.

[192] Aliaksandr Siarohin, Stéphane Lathuilière, Sergey Tulyakov, Elisa Ricci, and Nicu Sebe. “Animating arbitrary objects via deep motion transfer”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 2377–2386.

[193] Aliaksandr Siarohin, Stéphane Lathuilière, Sergey Tulyakov, Elisa Ricci, and Nicu Sebe. “First order motion model for image animation”. In: *Advances in neural information processing systems* 32 (2019).

[194] Aliaksandr Siarohin, Oliver J Woodford, Jian Ren, Menglei Chai, and Sergey Tulyakov. “Motion representations for articulated animation”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 13653–13662.

[195] Vibhaalakshmi Sivaraman, Pantea Karimi, Vedantha Venkatapathy, Mehrdad Khani, Sadjad Fouladi, Mohammad Alizadeh, Frédo Durand, and Vivienne Sze. “Gemino: Practical and robust neural compression for video conferencing”. In: *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. 2024, pp. 569–590.

[196] Jon Sneyers and Pieter Wuille. “FLIF: Free lossless image format based on MANIAC compression”. In: *2016 IEEE international conference on image processing (ICIP)*. IEEE. 2016, pp. 66–70.

[197] Iraj Sodagar. “The mpeg-dash standard for multimedia streaming over the internet”. In: *IEEE multimedia* 18.4 (2011), pp. 62–67.

[198] *SPH3D-GCN*. https://github.com/hlei-ziyan/SPH3D-GCN.

[199] Randall Stewart and Christopher Metz. “SCTP: new transport protocol for TCP/IP”. In: *IEEE Internet Computing* 5.6 (2001), pp. 64–69.

[200] Thomas Stockhammer and Miska M Hannuksela. “H. 264/AVC video for wireless transmission”. In: *IEEE Wireless communications* 12.4 (2005), pp. 6–13.

[201] Sanjib Sur, Ioannis Pefkianakis, Xinyu Zhang, and Kyu-Han Kim. “WiFi-assisted 60 GHz wireless networks”. In: *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking*. 2017, pp. 28–41.

[202] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. “Sequence to sequence learning with neural networks”. In: *Advances in neural information processing systems* 27 (2014).

[203] Vivienne Sze, Madhukar Budagavi, and Gary J Sullivan. “High efficiency video coding (HEVC)”. In: *Integrated circuit and systems, algorithms and architectures*. Vol. 39. Springer, 2014, p. 40.

[204] Zhaowei Tan, Jinghao Zhao, Yuanjie Li, Yifei Xu, and Songwu Lu. “{Device-Based}{LTE} Latency Reduction at the Application Layer”. In: *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. 2021, pp. 471–486.

[205] Steven L Tanimoto, Alon Itai, and Michael Rodeh. “Some matching problems for bipartite graphs”. In: *Journal of the ACM (JACM)* 25.4 (1978), pp. 517–525.

[206] *TensorFlow 1.14*. https://github.com/tensorflow/tensorflow/tree/r1.14.

[207] *The Octree Data Structure*. https://en.wikipedia.org/wiki/Octree.

[208] Nikolaos Thomos, Nikolaos V Boulgouris, and Michael G Strintzis. “Optimized transmission of JPEG2000 streams over wireless channels”. In: *IEEE Transactions on image processing* 15.1 (2005), pp. 54–67.

[209] Shao-Qin Tong, Han Bao, Jian-Cong Li, Ling Yang, Hou-Ji Zhou, Yi Li, and Xiang-Shui Miao. “Energy-Efficient Brain Floating Point Convolutional Neural Network Using Memristors”. In: *IEEE Transactions on Electron Devices* (2024).

[210] *TorchScript*. https://pytorch.org/docs/stable/jit.html.

[211] TP Link. *TP-Link Archer A7*. https://www.tp-link.com/us/home-networking/wifi-router/archer-a7. 2024.

[212] *Tp-link Archer A7*. https://www.tp-link.com/us/home-networking/wifi-router/archer-a7.

[213] UChi-JCL. *Grace*. Accessed: 2025-01-30. 2025. URL: https://github.com/UChi-JCL/Grace.

[214] *Unsplash Dataset*. unsplash.com/data.

[215] Subhashini Venugopalan, Marcus Rohrbach, Jeffrey Donahue, Raymond Mooney, Trevor Darrell, and Kate Saenko. “Sequence to sequence-video to text”. In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 4534–4542.

[216] *Video Multimethod Assessment Fusion*. https://en.wikipedia.org/wiki/Video_Multimethod_Assessment_Fusion. 2016.

[217] Irene Viola, Shishir Subramanyam, and Pablo Cesar. “A color-based objective quality metric for point cloud contents”. In: *Proceedings of the 12th International Conference on Quality of Multimedia Experience (QoMEX)*. 2020.

[218] *VIVE Cosmos Series*. https://www.vive.com/us/product/#cosmos%20series.

[219] *VIVE Pro Series*. https://www.vive.com/us/product/#pro%20series.

[220] *VIVE Tracker 3.0*. https://www.vive.com/us/accessory/tracker3.

[221] *VIVE Wireless Adapter*. https://www.vive.com/us/accessory/wireless-adapter.

[222] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. "Scaled-YOLOv4: Scaling Cross Stage Partial Network". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2021, pp. 13029–13038.

[223] Haiping Wang, Zhenhua Yu, Ruixiao Zhang, Siping Tao, Hebin Yu, and Shu Shi. "TwinStar: A Practical Multi-path Transmission Framework for Ultra-Low Latency Video Delivery". In: *Proceedings of the 31st ACM International Conference on Multimedia*. 2023, pp. 9234–9242.

[224] Pin-Chun Wang, Ching-Ling Fan, Chun-Ying Huang, Kuan-Ta Chen, and Cheng-Hsin Hsu. "Towards ultra-low-bitrate video conferencing using facial landmarks". In: *Proceedings of the 24th ACM international conference on Multimedia*. 2016, pp. 561–565.

[225] Shibo Wang, Shusen Yang, Hailiang Li, Xiaodan Zhang, Chen Zhou, Chenren Xu, Feng Qian, Nanbin Wang, and Zongben Xu. "SalientVR: saliency-driven mobile 360-degree video streaming with gaze information". In: *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking*. 2022, pp. 542–555.

[226] Song Wang, Jingqi Huang, and Xinyu Zhang. "Demystifying millimeter-wave V2X: Towards robust and efficient directional connectivity under high mobility". In: *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*. 2020.

[227] Ting-Chun Wang, Arun Mallya, and Ming-Yu Liu. "One-shot free-view neural talking-head synthesis for video conferencing". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2021, pp. 10039–10049.

[228] Yao Wang and Qin-Fan Zhu. "Error control and concealment for video communication: A review". In: *Proceedings of the IEEE* 86.5 (1998), pp. 974–997.

[229] Yifan Wang, Shihao Wu, Hui Huang, Daniel Cohen-Or, and Olga Sorkine-Hornung. "Patch-based Progressive 3D Point Set Upsampling". In: *Proceedings of CVPR*. 2019.

[230] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. "Image Quality Assessment: From Error Visibility to Structural Similarity". In: *IEEE Transactions on Image Processing* 13.4 (2004), pp. 600–612.

[231] *WikiArt dataset*. https://huggingface.co/datasets/huggan/wikiart/.

[232] Clark Wissler. "The Spearman correlation formula". In: *Science* 22.558 (1905), pp. 309–311.

[233] Timothy Woodford, Xinyu Zhang, Eugene Chai, Karthikeyan Sundaresan, and Amir Khojastepour. "Spacebeam: Lidar-driven one-shot mmwave beam management". In: *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*. 2021, pp. 389–401.

[234] Huikai Wu, Junge Zhang, and Kaiqi Huang. "Point Cloud Super Resolution with Adversarial Residual Graph Networks". In: *arXiv preprint arXiv:1908.02111*. 2019.

[235] Jiyan Wu, Chau Yuen, Bo Cheng, Ming Wang, and Junliang Chen. "Streaming high-quality mobile video with multipath TCP in heterogeneous wireless networks". In: *IEEE Transactions on Mobile Computing* 15.9 (2015), pp. 2345–2361.

[236] Junda Wu, Haoliang Wang, Tong Yu, Gang Wu, Stefano Petrangeli, Handong Zhao, Sungchul Kim, and Viswanathan Swaminathan. "Content-aware Progressive Image Compression and Syncing". In: *2023 IEEE International Symposium on Multimedia (ISM)*. IEEE. 2023, pp. 221–224.

[237] Zongshen Wu, Chin-Ya Huang, and Parameswaran Ramanathan. "Measuring Millimeter Wave based Link Bandwidth Fluctuations during Indoor Immersive Experience". In: *IEEE Networking Letters* (2022).

[238] Guiyu Xia, Huaijiang Sun, Beijia Chen, Qingshan Liu, Lei Feng, Guoqing Zhang, and Renlong Hang. "Nonlinear low-rank matrix completion for human motion recovery". In: *IEEE Transactions on Image Processing* 27.6 (2018), pp. 3011–3024.

[239] Chenhao Xie, Xie Li, Yang Hu, Huwan Peng, Michael Taylor, and Shuaiwen Leon Song. "Q-vr: system-level design for future mobile collaborative virtual reality". In: *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 2021, pp. 587–599.

[240] Ruolin Xing, Mengwei Xu, Ao Zhou, Qing Li, Yiran Zhang, Feng Qian, and Shangguang Wang. "Deciphering the enigma of satellite computing with cots devices: Measurement and analysis". In: *arXiv preprint arXiv:2401.03435* (2024).

[241] Tan Xu, Bo Han, and Feng Qian. "Analyzing viewport prediction under different VR interactions". In: *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*. 2019, pp. 165–171.

[242] Zhongcong Xu, Jianfeng Zhang, Jun Hao Liew, Hanshu Yan, Jia-Wei Liu, Chenxu Zhang, Jiashi Feng, and Mike Zheng Shou. “Magicanimate: Temporally consistent human image animation using diffusion model”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2024, pp. 1481–1490.

[243] Francis Y. Yan, Hudson Ayers, Chenzhi Zhu, Sadjad Fouladi, James Hong, Keyi Zhang, Philip Levis, and Keith Winstein. “Learning in situ: a randomized experiment in video streaming”. In: *Proceedings of USENIX NSDI*. 2020.

[244] Jianchao Yang, John Wright, Thomas S. Huang, and Yi Ma. “Image Super-Resolution Via Sparse Representation”. In: *IEEE Transactions on Image Processing* 19.11 (2010), pp. 2861–2873.

[245] Hyunho Yeo, Chan Ju Chong, Youngmok Jung, Juncheol Ye, and Dongsu Han. “NEMO: Enabling Neural-enhanced Video Streaming on Commodity Mobile Devices”. In: *Proceedings of ACM MobiCom*. 2020.

[246] Hyunho Yeo, Sunghyun Do, and Dongsu Han. “How will Deep Learning Change Internet Video Delivery?” In: *Proceedings of ACM HotNets*. 2017.

[247] Hyunho Yeo, Youngmok Jung, Jaehong Kim, Jinwoo Shin, and Dongsu Han. “Neural Adaptive Content-aware Internet Video Delivery”. In: *Proceedings of USENIX OSDI*. 2018.

[248] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. “A Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP”. In: *Proceedings of ACM SIGCOMM*. 2015.

[249] Lequan Yu, Xianzhi Li, Chi-Wing Fu, Daniel Cohen-Or, and Pheng-Ann Heng. “PU-Net: Point Cloud Upsampling Network”. In: *Proceedings of CVPR*. 2018.

[250] Zhenhui Yuan, Hrishikesh Venkataraman, and Gabriel-Miro Muntean. “iBE: A novel bandwidth estimation algorithm for multimedia services over IEEE 802.11 wireless networks”. In: *IFIP/IEEE International Conference on Management of Multimedia Networks and Services*. Springer. 2009, pp. 69–80.

[251] *ZED 2i Camera*. https://www.stereolabs.com/zed-2i.

[252] *ZED 3D Object Detection*. https://www.stereolabs.com/docs/object-detection/.

[253] *zed-openpose Codebase*. https://github.com/stereolabs/zed-openpose.

[254] Anlan Zhang, Chendong Wang, Bo Han, and Feng Qian. “{YuZu}:{Neural-Enhanced} Volumetric Video Streaming”. In: *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 2022, pp. 137–154.

[255] Anlan Zhang, Chendong Wang, Bo Han, and Feng Qian. “Efficient volumetric video streaming through super resolution”. In: *Proceedings of the 22nd International Workshop on Mobile Computing Systems and Applications*. 2021, pp. 106–111.

[256] Anlan Zhang, Chendong Wang, Yuming Hu, Ahmad Hassan, Zejun Zhang, Bo Han, Feng Qian, and Shichang Xu. “Habitus: boosting mobile immersive content delivery through full-body pose tracking and multipath networking”. In: *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. 2024, pp. 1677–1695.

[257] Anlan Zhang, Chendong Wang, Xing Liu, Bo Han, and Feng Qian. “Mobile volumetric video streaming enhanced by super resolution”. In: *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*. 2020, pp. 462–463.

[258] Ding Zhang, Puqi Zhou, Bo Han, and Parth Pathak. “M5: Facilitating Multi-user Volumetric Content Delivery with Multi-lobe Multicast over mmWave”. In: (2022).

[259] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. “The unreasonable effectiveness of deep features as a perceptual metric”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 586–595.

[260] Wenxiao Zhang, Feng Qian, Bo Han, and Pan Hui. “Deepvista: 16k panoramic cinema on your mobile device”. In: *Proceedings of the Web Conference 2021*. 2021, pp. 2232–2244.

[261] Xumiao Zhang, Anlan Zhang, Jiachen Sun, Xiao Zhu, Y Ethan Guo, Feng Qian, and Z Morley Mao. “Emp: Edge-assisted multi-vehicle perception”. In: *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*. 2021, pp. 545–558.

[262] Zhimeng Zhang, Lincheng Li, Yu Ding, and Changjie Fan. “Flow-Guided One-Shot Talking Face Generation With a High-Resolution Audio-Visual Dataset”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 3661–3670.

[263] Peijun Zhao, Chris Xiaoxuan Lu, Jianan Wang, Changhao Chen, Wei Wang, Niki Trigoni, and Andrew Markham. “mid: Tracking and identifying people with millimeter wave radar”. In: *2019 15th International Conference on Distributed Computing in Sensor Systems (DCOSS)*. IEEE. 2019, pp. 33–40.

[264] Xiao Zhu, Jiachen Sun, Xumiao Zhang, Y Ethan Guo, Feng Qian, and Z Morley Mao. “MPBond: efficient network-level collaboration among personal mobile devices”. In: *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*. 2020, pp. 364–376.

[265] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. “A comprehensive survey on transfer learning”. In: *Proceedings of the IEEE* 109.1 (2020), pp. 43–76.

[266] *zlib 1.3.1 Release.* https://github.com/madler/zlib/archive/refs/tags/v1.3.1.zip.

[267] *Zoom system requirements: Windows, macOS, Linux.* https://support.zoom.com/hc/en/article?id=zm_kb&sysparm_article=KB0060748.

[268] *Zoom: One platform to connect.* https://zoom.us/. 2024.