Resource-Efficient Video Analytics and Streaming for Immersive Experiences

A thesis submitted in fulfilment of the requirements
for the degree of

Doctor of Philosophy
in
Information Systems

by

Jingrou Wu

to

School of Computer Science
Faculty of Engineering and Information Technology
University of Technology Sydney
NSW - 2007, Australia

February 2025

CERTIFICATE OF ORIGINAL AUTHORSHIP

I, *Jingrou Wu*, declare that this thesis is submitted in fulfilment of the requirements for the award of Doctor of Philosophy, in the *School of Computer Science*, *Faculty of Engineering and Information Technology* at the University of Technology Sydney, Australia.

This thesis is wholly my own work unless otherwise referenced or acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis.

I certify that the work in this thesis has not previously been submitted for a degree nor has it been submitted as part of the requirements for a degree at any other academic institution except as fully acknowledged within the text. This thesis is the result of a Collaborative Doctoral Research Degree program with the Southern University of Science and Technology.

This research is supported by the Australian Government Research Training Program.

SIGNATURE: Production Note:
Signature removed prior to publication.

[Jingrou Wu]

DATE: 10th February, 2025

PLACE: Sydney, Australia

Abstract

The evolution of immersive media technologies, such as Extended Reality (XR) and volumetric video, has gained significant attention from academia and industry. Video analytics, powered by artificial intelligence, plays an essential role in immersive applications. It facilitates real-time object detection, tracking, and scene understanding, thereby enabling interactive experiences. Real-time volumetric video streaming emerges as a critical application in immersive media. It offers 6 Degrees of Freedom (DoF), allowing users to explore the video entirely by changing their positions (X, Y, Z) and viewport directions (yaw, pitch, roll). Despite their promise, immersive media applications demand extensive computation and bandwidth resources, often exceeding the capabilities of current infrastructures. This thesis addresses these challenges by proposing dynamic resource allocation strategies and efficient streaming schemes to provide high-quality, low-latency real-time analytics and video streaming for immersive media. The proposed solutions aim to optimize video analytics and volumetric video streaming within limited resources, ensuring a seamless and responsive user experience.

First, the thesis introduces a dual-image Field Programmable Gate Arrays (FPGAs) solution in video analytics pipelines to address the limitations of immutable computing resources. This innovation allows for flexible resource allocation between CPUs and GPUs by switching between different images of FPGA. Novel algorithms for FPGA resource allocation and video analytics configuration selection are developed to optimize

accuracy across both single and multiple-camera scenarios. Second, the thesis proposes a Progressive Layer-based Volumetric Video Streaming (PLVS) framework to overcome the challenge of limited bandwidth resources and network dynamics by prefetching future frames in advance when current bandwidth resources are sufficient. To further improve network utilization, volumetric videos are partitioned into several layers of varying quality, allowing for progressive refinement and efficient retrieval during streaming. The PLVS framework is applied to two practical scenarios where frames can be skipped or not. It enables adaptive quality refinement and more efficient video prefetching in diverse streaming environments. Third, the thesis develops a Hierarchical Reinforcement Learning-based (HRL) streaming scheme to balance computing and bandwidth resources in tile-based volumetric video streaming. Tiling and culling techniques are employed to reduce the transmission data size by eliminating invisible tiles, which bring extra computation overhead. The culling performance and compression efficiency are highly dependent on tile size, and the optimal tile size varies depending on the viewports of users and dynamic bandwidth conditions. The proposed streaming scheme jointly considers the tile size selection and quality allocation, maintaining a balance between computing and bandwidth resources.

Keywords: Immersive media, Video analytics, Volumetric video, Resource allocation, Configuration optimization

Dissertation directed by A/Prof. Jing Jiang, A/Prof. Jin Zhang, A/Prof. Guodong Long, and Dist/Prof. Chengqi Zhang

Australian Artificial Intelligence Institute

Faculty of Engineering and IT

University of Technology Sydney

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my principal supervisor Professor Jing Jiang, and co-supervisors Professor Jin Zhang, Professor Chengqi Zhang, and Professor Guodong Long. Their invaluable guidance and selfless support have been instrumental throughout my PhD years. Professor Jing Jiang helped me navigate my PhD, offering continuous mentorship at every stage. She provided clear advice on my research plan, kept my work on track, and guided me through key moments, ensuring steady progress toward my thesis. Professor Jin Zhang has been a supportive and trusted mentor not only in my research but also in my personal well-being. Her academic guidance helped me develop critical thinking and grow into an independent researcher. Beyond academia, her kindness, patience, and encouragement gave me the strength to overcome difficulties and stay motivated. Professor Chengqi Zhang and Professor Guodong Long inspired and supported me with their profound expertise and rigorous academic attitude, encouraging me to strive for higher standards in my own work. Additionally, I would like to extend my sincere gratitude to Professor Dan Wang, from the Hong Kong Polytechnic University, for his generous mentorship and insightful discussions during our collaboration. His rich research experience has shaped my thinking and approach to research and his expertise in algorithm optimization has strengthened the theoretical foundation of my work.

I am also deeply grateful to the senior PhD students and postdocs, Dr. Chuang Hu, Dr. Mingzhe Li, Dr. Bai Yan, and Dr. Jiao Li, who have generously shared their knowledge

and experience with me. They guided me through challenges in my research, refined my ideas, and improved my academic writing. Besides, I appreciate my colleagues and group mates, Rui Lu, Xiaojun Wang, Haoxian Liu, ..., whose engaging discussions and collaboration have greatly improved my work. Their support has made my time in the lab both productive and enjoyable. Beyond academics, I am grateful to my friends, especially my dearest friend, TianXin Lu, who has been a constant source of encouragement and emotional support in my life. From casual conversations to shared meals and basketball games, their companionship has made this journey more enjoyable and fulfilling. I would also like to express my gratitude to the University of Technology Sydney and the Southern University of Science and Technology for funding my PhD research and providing essential support throughout my studies.

Lastly, I sincerely thank my parents for their unconditional love and endless support. Though they are unfamiliar with my research, they have always believed in me, even during the toughest times. They have always encouraged me to pursue what I truly admire, while lighting my way home, no matter how far I go. Without them, I would not be where I am today, nor would I have achieved what I have.

Jingrou Wu

Sydney, Australia, 2025

List of Publications

Related to the Thesis :

1. **Jingrou Wu**, Chuang Hu, Jin Zhang, Dan Wang, Jing Jiang, *Gemini+: Enhancing Real-Time Video Analytics with Dual-Image FPGAs*. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. Early Access.

2. **Jingrou Wu**, Haoxian Liu, Jin Zhang, Dan Wang, Jing Jiang, *P^2VS: Progressive Partition-based Volumetric Video Streaming under Network Dynamics*. ACM International Conference on Multimedia. Accept.

3. **Jingrou Wu**, Jin Zhang, Dan Wang, Jing Jiang, *Layer-based Prefetching for Network Dynamics in Volumetric Video Streaming*. IEEE Transactions on Circuits and Systems for Video Technology. Under Review.

4. **Jingrou Wu**, Jin Zhang, Xiaojun Wang, Dan Wang, Jing Jiang, *Tile Configuration for QoE-Aware Volumetric Video Streaming via Hierarchical Reinforcement Learning*. ACM Transactions on Sensor Networks. Major Revison.

Others :

5. **Jingrou Wu**, Jin Zhang, *Matching with Externalities for Device Assignment in IoT Sharding Blockchain*. Edge Computing and IoT: Systems, Management and Security, 2022.

6. **JINGROU WU**, JIN ZHANG, *Committee Selection Based on Game Theory in Sharding Blockchain*. Edge Computing and IoT: Systems, Management and Security, 2022.

7. MINGZHE LI, **JINGROU WU**, WEI WANG. JIN ZHANG, *Toward Privacy-Preserving Task Assignment for Fully Distributed Spatial Crowdsourcing*. IEEE Internet of Things Journal, vol. 8, no. 18, pp. 13991-14002, 2021

Table of Contents

List of Figures

List of Tables

CHAPTER 1

INTRODUCTION

1.1 Research Background

Immersive media applications such as Extended Reality (XR) and volumetric video have become increasingly popular nowadays due to the rapid development of wireless technology and smart mobile devices. These applications provide hyper-realistic experiences that blur the lines between virtual and physical worlds and are widely used in entertainment, education, and health care.

Video analytics and immersive video streaming are two essential technologies in immersive applications. Video analytics facilitates immersive experiences by enabling intelligent processing of visual data, while immersive video streaming delivers high-quality immersive content to users in real time. Among various immersive video formats, volumetric video stands out for its ability to capture and render dynamic 3D representations, providing a more realistic and interactive experience. However, both technologies face challenges of limited resources, requiring efficient optimization strategies.

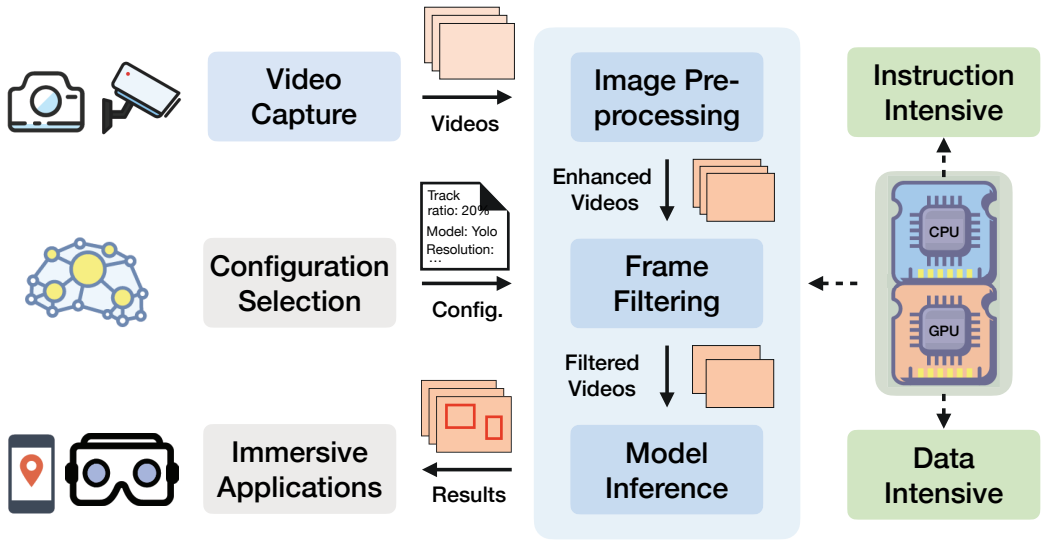


Figure 1.1: Overview of the video analytics pipeline.

1.1.1 Video Analytics

Video analytics extracts and analyzes important features from videos using computer vision. It contains multiple tasks such as object detection, shape recognition, and motion tracking. These tasks are executed through a video analytics pipeline that transforms raw video inputs into meaningful analytics results.

A video analytics pipeline typically encompasses the entire process from video capture to video analysis. While its design may vary depending on the specific requirements of different analytics tasks, most video analytics pipelines share a common structure, consisting of several key stages, as illustrated in Fig. 1.1.

Video capture. The pipeline begins with video capture from cameras or other recording devices. These raw video frames are then passed into the following modules for further processing.

Image pre-processing. Image pre-processing aims to enhance image quality for further feature extraction and model inference. This stage includes various techniques

such as noise reduction [53], image enhancement [79], scaling and resizing [76]. Recently, some smart cameras [90] have integrated these algorithms. They enable automatic image pre-processing based on different capture environments, thereby improving the efficiency and accuracy of subsequent video analytics tasks.

Frame filtering. Frame filtering aims to reduce the computational load of model inference by eliminating irrelevant frames. The primary method used in frame filtering is frame differencing, which can be applied directly to pixels or other low-level features such as edges and corners. Recently, machine learning has been applied in frame filtering. A simple binary classification model [12, 46] is trained to detect whether the frame contains target objects or not. Instead of filtering out the frame directly, background subtraction [25] extracts foreground objects and removes irrelevant background information. These filtering algorithms significantly contribute to reducing computational demands in video analytics.

Model inference. Model inference is a core component in the video analytics pipeline. It utilizes state-of-the-art Deep Neural Network (DNN) models, such as YOLO [84], RetinaNet [65], and ResNet [30] to perform specific video analytics tasks. Lightweight models are often integrated at this stage to reduce computational demands. For instance, traditional object-tracking algorithms can be employed to process frames with small differences. Overall, model inference plays a significant role in the video analytics pipeline by delivering final results. However, it consumes overwhelming computing resources to achieve high accuracy, which can become a bottleneck for real-time analytics.

A video analytics pipeline typically allows the configuration of key parameters, such as frame rate, resolution, and model selection, to balance computational demands and performance across diverse applications and environments. For example, it achieves high analytics accuracy with a high frame rate, high resolution, and complex DNN models.

However, it demands significant computing resources and results in high latency, which may be unacceptable for real-time applications run on resource-constrained devices. Fortunately, such resource-intensive configurations are not always necessary. For instance, tasks like detecting large, slow-moving objects can be effectively performed with lower frame rates, reduced resolution, and simpler DNN models without significant loss in accuracy. Therefore, it is essential to select configurations dynamically to optimize analytics accuracy and resource efficiency.

Meanwhile, different computing resources play specific roles within the video analytics pipeline, optimizing performance for various computational tasks. CPUs are good at instruction-intensive tasks involving control and sequential computing. This makes them suitable for frame filtering and lightweight tracking algorithms. In contrast, GPUs, designed for massively parallel computations, significantly accelerate data-intensive tasks such as DNN model inference. It is essential to effectively integrate heterogeneous computing resources in the video analytics pipeline, especially for devices with limited computing resources.

1.1.2 Volumetric Video Streaming

Volumetric video captures the three-dimensional information of a real-world scene using arrays of RGB-D cameras, e.g., Microsoft Kinect [6] and Intel RealSense [5]. These devices capture both color (RGB) and depth (D) data, enabling the creation of high-fidelity 3D models from multiple viewpoints. The process involves complex steps, including camera calibration, bias correction, and background subtraction to integrate RGB-D frames into a unified 3D model [16]. Most volumetric videos are created in studios such as 8i [2], 4DViews [1], and Microsoft Mixed Reality Capture Studio [7]. Recently, sports events such as the NBA have been interested in live volumetric video by installing hundreds of cameras around the court.

Unlike the 2D video, which is composed of pixel arrays, volumetric video contains a sequence of 3D models. These 3D models are represented by meshes or point clouds (PtCls) or other 3D formats. Among these, the point cloud format is particularly popular due to its simplicity and flexibility and is widely employed in various volumetric videos. A PtCl consists of a set of points with 3D coordinates and attributes such as colors and intensity. This thesis specifically focuses on PtCl-based volumetric videos as well.

The 3D formats in volumetric video provide a Six Degrees of Freedom (DoF) movement, allowing viewers to change their positions (X, Y, Z) and viewport directions (yaw, pitch, roll) to watch the video from any angle, offering a more immersive and engaging experience. However, volumetric video streaming requires high transmission bandwidth due to the larger data size of 3D formats compared to 2D pixels. For example, the size of a single frame with 750 K 3D points is 17 MB. The bandwidth for a 30 frames per second video without compression is up to 4 Gbps, which is beyond most existing network conditions.

Various volumetric video compression algorithms have been proposed to reduce the extremely large data size. Typically, there are two kinds of PtCl-based volumetric video compression algorithms: video-based compression and geometry-based compression. The main idea of video-based compression such as V-PCC [44] is to project the volumetric video into 2D images and then leverage the existing 2D video codecs to encode. It significantly reduces data size but with high encoding and decoding overhead. In contrast, geometry-based compression encodes PtCl directly with efficient structure representation such as octree [54, 88] and k-d tree. Though it does not achieve as high compression efficiency as video-based compression, the compression overhead is much smaller. In this work, Google Draco [4] is adopted as the compression tool because of its low computational complexity and real-time decoding capability, making it well-suited for streaming.

Although compression techniques reduce the size of volumetric video data, they alone are insufficient to address the network dynamics and bandwidth resource limitations in volumetric video streaming. Frequent bandwidth fluctuations can lead to abrupt quality switches and playback stalls, degrading the immersive experience. Additionally, limited bandwidth resources may decrease the video quality and increase rebuffering frequency, further impairing the quality of experience (QoE).

To mitigate the impact of network fluctuations, prefetching is commonly applied in streaming systems. It ensures smoother playback and reduces buffering delays by retrieving future video frames in advance when bandwidth resources are sufficient. Prefetching strategies have been extensively explored in conventional pixel-based video streaming and are increasingly applied to volumetric video to enhance playback stability.

Beyond network dynamics, volumetric video streaming is also constrained by limited bandwidth resources. Since users typically focus on the content within their field of view (FoV), there is still significant room to further reduce streaming data size. To achieve this, tiling partitions volumetric video into smaller spatial segments, allowing selective transmission of viewport-relevant content, while culling determines which tiles are necessary for transmission under different viewports. Specifically, frustum culling [11] removes tiles outside the users' FoV, while occlusion culling [73] eliminates tiles that are obstructed by other tiles. These techniques ensure that only relevant volumetric data is transmitted, effectively reducing bandwidth consumption while maintaining an immersive viewing experience.

1.2 Research Problem

While immersive media technologies hold great potential across diverse fields, they demand high computing and bandwidth resources to support high-quality and low-latency

video analytics and volumetric video streaming tasks. It is particularly significant in real-world use cases such as telemedicine, industrial maintenance, virtual collaboration, and live immersive entertainment, where users expect seamless, responsive, and high-quality experiences. In these domains, even minor delays or drops in video quality can compromise user experience, decision-making, or analytical accuracy. Hence, developing effective solutions to ensure low-latency, high-quality immersive media delivery under resource constraints has strong practical relevance. However, existing solutions often fail to efficiently utilize these resources, leaving room for improvement and refinement.

This thesis is motivated by the key observations that the computing and communication resources are not fully exploited by existing immersive applications. Many current systems rely on static or suboptimal strategies that overlook the dynamic nature of workloads and network conditions. As a result, there is a mismatch between system capabilities and application demands, leading to inefficiencies and degraded user experiences. Therefore, this thesis aims to address the following research problems in resource-efficient video streaming and analytics:

Research problem 1: *How can we fully utilize heterogeneous computing resources in the video analytics pipeline?*

Video analytics pipelines rely on heterogeneous computing resources, including CPUs for control-intensive tasks and GPUs for data-intensive tasks. Existing video analytics systems typically assume that CPUs and GPUs are independent and fixed. Therefore, they are not able to transform into each other during run time. However, this rigidity limits performance because of an imbalance between CPU-GPU workloads and computing resources. As a result, one resource may be overloaded while the other remains underutilized, reducing overall system efficiency. A video analytics pipeline with flexible computing resources is needed to overcome these challenges.

Research problem 2: *How can we mitigate the impact of network dynamics in volumetric video streaming?*

Prefetching is a widely used strategy to counteract network fluctuations by downloading future video segments in advance when bandwidth conditions allow. However, straightforward prefetching methods introduce a critical trade-off between video quality and bandwidth efficiency. If prefetched segments cannot be enhanced later, the overall video quality is compromised. Conversely, if retransmissions are enabled to improve quality, previously transmitted frames may be discarded, leading to bandwidth wastage. Addressing this challenge requires a more adaptive prefetching approach that can maintain high video quality while minimizing bandwidth waste.

Research problem 3: *How can we balance the computing resources and bandwidth resources in tile-based volumetric video streaming?*

Tiling and culling are commonly used to reduce volumetric video streaming data size by eliminating invisible tiles. While this approach optimizes bandwidth usage, it introduces additional computational overhead for performing culling. The tile size significantly impacts the trade-off between the two resources. For example, smaller tiles increase culling accuracy by reducing redundant data, while larger tiles may retain more invisible content. However, it is impractical to set the tile size as small as possible because it leads to extremely high computational overhead due to the increased number of tiles to process. Moreover, the optimal tile size is highly dependent on bandwidth conditions and user viewports, making a static configuration unsuitable for real-world streaming scenarios. These limitations motivate the need for tile size adaptation in volumetric video streaming.

To address these challenges, this thesis proposes three key solutions: 1) A video analytics pipeline with flexible computing resources that dynamically reconfigures CPU-

GPU resources to adapt to diverse workloads, improving video analytics performance. 2) A layer-based prefetching strategy for volumetric video streaming, which enhances video quality while minimizing bandwidth wastage under network dynamics. 3) A tile-based volumetric video streaming framework with flexible tile size that balances computing and bandwidth resources, improving streaming efficiency under limited computing and bandwidth resources. These solutions provide a comprehensive approach to improving resource efficiency in immersive media applications.

1.3 Research Contribution

This thesis aims to address the challenges of limited computing and bandwidth resources in immersive media applications by proposing novel strategies in video analytics and volumetric video streaming. The key contributions are as follows:

- This research leverages the dual-image FPGAs to propose a video analytics pipeline with flexible computing resources. This approach enables dynamic CPU-GPU resource partitioning to adapt to varying CPU-GPU workloads, improving video analytics performance under limited and fixed computing resources.
- This research designs a layer-based prefetching scheme for volumetric video streaming under network dynamics. This approach improves video quality by enabling the refinement of transmitted segments. Furthermore, it minimizes bandwidth wastage by transmitting additional content and integrating it with previously received data.
- This research develops a tile-based volumetric video streaming framework with flexible tile size to balance between limited computing resources and bandwidth

resources. It dynamically adjusts tile sizes and determines tile quality in response to network conditions and viewport changes.

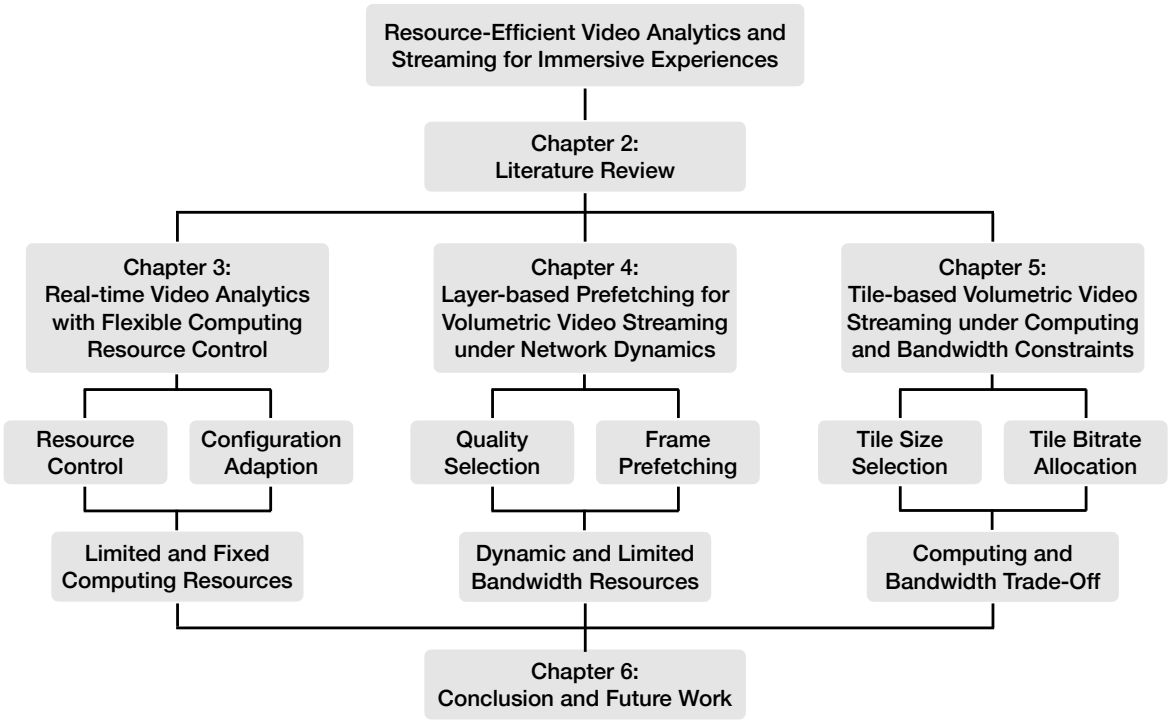


Figure 1.2: The organization of this thesis.

1.4 Thesis Organization

This thesis seeks to enhance the immersive media experience within the constraints of limited resources. It introduces novel strategies to optimize video analytics and volumetric video streaming. The thesis is structured into six chapters, as illustrated in Fig. 1.2, with each chapter addressing a different aspect of the research:

- Chapter 1: This chapter provides an overview of the research background, highlighting the significance of immersive media technologies and the challenges associated

with video analytics and volumetric video streaming. It also outlines three research problems and the contributions of this research.

- Chapter 2: This chapter offers a comprehensive literature review related to video analytics and volumetric video streaming.

- Chapter 3: This chapter introduces a flexible computing resources design in video analytics pipelines to address the inefficient utilization of limited computing resources. It investigates the CPU-GPU resource control and configuration adaptation for real-time video analytics. Two problems of practical importance, a single-camera and multi-camera dual computing resource control problem, are formulated and analyzed, with proposed algorithms to optimize resource control and configuration selection.

- Chapter 4: This chapter presents a novel layer-based prefetching scheme aimed at overcoming bandwidth limitations and network dynamics in volumetric video streaming. The method integrates layer partitioning with volumetric video compression to support progressive streaming. The approach jointly optimizes video quality and prefetching strategy to enhance streaming performance, considering two volumetric video scenarios: allowing or forbidding frame skips.

- Chapter 5: This chapter focuses on the balance between computing and bandwidth resources in tile-based volumetric video streaming. It formulates the tile size selection and bitrate allocation (tile quality selection) as a Markov Decision Process(MDP) problem to optimize the long-term QoE. An HRL-based streaming scheme is proposed to adapt to dynamic viewports, video content, and bandwidth conditions.

- Chapter 6: This chapter summarizes the thesis and discusses future research to enhance resource efficiency in immersive media technologies.

CHAPTER 2

LITERATURE REVIEW

The following literature review provides a brief introduction to two topics that are closely related to the research: video analytics and volumetric video streaming.

2.1 Video Analytics

In this section, we review the existing literature on video analytics with different computing resources. Video analytics pipelines are crucial for processing and analyzing video data in real time and are core components in immersive media applications such as XR. These pipelines can utilize either homogeneous or heterogeneous computing resources, each offering distinct advantages and challenges.

2.1.1 Video Analytics with Homogeneous Computing Resources

Homogeneous computing resources refer to identical processors or cores. In video analytics, these typically are GPUs and dedicated AI chips such as TPU [110]. These computing resources are crucial for handling the heavy computational and memory

demands of Deep Neural Network (DNN) or Convolutional Neural Network (CNN) models, such as AlexNet [51], RCNN [26], and GoogleNet [96]. To support video analytics on devices with limited computing resources, different efforts have been made.

Neural Network Model Modification and Acceleration. Several works address the challenge by directly reducing model complexity and/or improving model efficiency. Approaches like SqueezeNet [42] and Deep Compression [28] focus on reducing memory usage, while others such as XNorNet [82] and MobileNet [34] aim to speed up model processing. Additionally, hardware accelerators including commercial products like Google's TPU [18] and NVIDIA's SCNN [74], as well as FPGAs [23], have shown promise in enhancing the efficiency of these models. Traditional feature extraction methods in computer vision are accelerated in FPGA [103]. Furthermore, Deepburning [111] proposes a design automation tool to build different NN models with FPGAs.

Computation offloading. Computation offloading empowers the resource-constrained devices by offloading the full or partial video analytics tasks to the server with powerful computation ability. Deepdecision [81] determines whether either process the video clip locally or remotely. It takes complex factors such as model accuracy, video quality, battery constraints, network data usage, and network conditions into consideration. Galanopoulos et al. [22] optimizes video analytics accuracy by jointly configuring the service and wireless network parameters with the Automated Machine Learning (AutoML) framework. JCAB [104] jointly considers configuration adaptation and bandwidth allocation in a multi-user system. CASVA [126] proposes a deep reinforcement learning (DRL) model to configure frame rate, resolution, and quantization parameters in video analytics. The DRL approach does not make any assumptions about the environment but to learn about it. The authors in [112] investigate adaptive configurations for serverless-based video analytics by determining which key frames should be offloaded to the serverless platform.

Model Partition and Offloading. Model partition and offloading aim to split the complex DNN model into parts that can be processed separately on devices and servers. Neurosurgeon [48] proposes a lightweight scheduler to automatically partition DNN computation at the layer level. It adapts to various DNN architectures at runtime because of the predicting model for latency and power consumption estimation. DADS [35] explores the partition of DNN models with a directed acyclic graph (DAG) structure instead of a chain-based model in Neurosurgeon. It also proposes a dynamic partition scheme based on network conditions. To further reduce network latency, JALAD [55] introduces in-layer compression technology in DNN partition. Recent works extend these partitioning strategies across more complex network structures involving multiple nodes. DDNN [99] and Lin et al. [63] consider the DNN partition in three tiers: end users, the edge server, and the cloud server. DINA [70] considers a network structure with multiple edge nodes. It first partitions DNN models into several (sub)layers and then presents a layer placement scheme based on the matching theory. The authors in [56] investigate fine-grained slicing of DNNs across edge devices, further optimizing distributed inference performance.

While the techniques mentioned above greatly enhance the performance of video analytics on devices with limited resources, they primarily focus on optimizing specific hardware components like GPUs or dedicated AI processors. This often leaves other available resources, such as CPUs, underutilized. It remains a challenge to integrate these resources effectively, indicating a potential area for further research and development to achieve a more complete use of all computing resources within devices.

2.1.2 Video Analytics with Heterogeneous Computing Resources

Heterogeneous computing resources consist of diverse types of processors, such as CPUs, GPUs, and dedicated AI accelerators. This approach leverages the unique

strengths of each component to handle different types of tasks within a video analytics pipeline, improving video analytics performance and efficiency.

Filter-based Video Analytics. Videos often contain numerous irrelevant frames without any target objects. It is practical to utilize CPUs to filter out these frames. FFS-VA [122, 123] specifically aims to filter out frames without target objects. The CPU in FFS-VA is assigned to remove frames containing only background, and the GPU handles object detection tasks. Reducto [62] introduces a lightweight on-camera filtering algorithm that dynamically adapts its filtering decisions based on the correlation between feature type, filtering threshold, query accuracy, and video content. RES [9] leverages the CPU to filter low-value stream objects using configurable rules, while high-value objects are processed by the GPU for identification. Crucio [129] proposes an end-to-end neural codec to coordinate batch-level filtering, accuracy-oriented intraframe, and inter-frame compression, aiming to reduce video analytics processing latency.

Tracking-based Video Analytics. Considering the temporal correlation of the video frames, tracking methods act as a good way to share the heavy workload with the DNN/CNN model, which is usually implemented on CPUs. MARLIN [10] proposes a sequential video analytics pipeline with an object detector and tracker. AdaVP [67] improves video analytics efficiency by a parallel detection and tracking pipeline. An online change detector threshold selection algorithm is studied in [29]. The frames beyond the threshold are offloaded to the server, while others are processed locally with the tracking module. VisFlow [116] investigates tracking-based video analytics on heterogeneous collaborative cameras. It adaptively increases detection frequency in response to tracking drift, thereby improving the accuracy of video analytics.

Adaptive CPU-GPU scheduling. Adaptive CPU-GPU scheduling is crucial in systems with multiple CPUs and GPUs. It assigns different numbers of CPUs and GPUs to

different video analytics tasks. DART [113] proposes a DNN scheduling framework that addresses the diverse resource requirements of individual DNN layers. This framework facilitates the co-utilization of CPUs and GPUs in inference job execution through effective resource allocation and scheduling strategies. LaLaRand [47] proposes a CPU-GPU scheduling algorithm for layer-based DNN model partition. This method employs CPU-friendly quantization to significantly reduce CPU processing latency, enabling CPUs to efficiently handle multiple DNN layers. Authors in [98] consider a job scheduling problem in a heterogeneous computing system and propose a CPU-GPU-utilization-aware and energy-efficient algorithm for it. Recent studies have focused on fine-grained coordination of tasks across heterogeneous resources to enhance overall throughput and responsiveness. RegenHance [108] proposes a region-based enhancement framework that selectively processes important video regions and employs a profile-based planner to dynamically allocate edge computing resources, improving video analytics throughput under limited computational budgets. Chen et al. [15] present a modular scheduling framework that decomposes inference tasks into components and assigns them across heterogeneous edge devices based on resource availability and user quality of service requirements. Yu et al. [119] introduce an intra-device, complexity-aware scheduling strategy that dynamically distributes DNN inference tasks between lightweight on-device accelerators and the GPU to optimize accuracy within latency constraints. At a higher level of granularity.

Although these works leverage the advantages of heterogeneous computing resources effectively, the immutable nature of these resources can sometimes limit video analytics performance, particularly when the workload mismatches with the available heterogeneous resources.

2.2 Volumetric Video Streaming

In this section, we review recent works in volumetric video streaming. These frameworks usually enhance QoE by dynamically adjusting video configurations based on available bandwidth and computing resources. We categorize related works into three primary types: frame-based streaming, tile-based streaming, and RL-based streaming approaches.

Frame-based volumetric video streaming. These approaches utilize a single frame as the minimal unit for rate adaptation, which simplifies implementation. DASH-PC [33] first extends the concept of dynamic adaptive streaming over HTTP (DASH) towards the volumetric video. It introduces adaptation sets for varying point cloud densities. PCC-DASH [102] specializes in handling multiple point cloud objects instead of a single object. It offers different compression versions for each object in the same frame. Furthermore, YuZu [120] leverages 3D super-resolution (SR) techniques to improve streaming efficiency while maintaining high video quality. This method allows for a strategic choice between transmitting lower-quality frames and enhancing them via neural networks or streaming high-quality frames directly. This technique effectively balances bandwidth with computing resources. VV-DASH [32] introduces a DASH-compatible format that groups independently decodable frames, allowing efficient adaptation and parallel decoding in volumetric streaming.

While these efforts have significantly contributed to rate adaptation in volumetric video streaming, they do not consider the fact that only a fraction of the content is visible to the viewer. Therefore, substantial bandwidth resources are wasted to transmit content that may not be necessary or relevant.

Tile-based volumetric video streaming. Tile-based streaming tackles the inefficiency of unnecessary data transmission by dividing the video into small tiles, each

of which can be streamed and adjusted independently based on viewer interaction and visibility. Park et al. [75] introduce 3D tiles to volumetric video, proposing a greedy algorithm for quality allocation to maximize video quality. ViVo [27] further enhances streaming efficiency with visibility-aware optimizations. The optimizations include removing non-visible tiles and reducing the quality of occluded or distant tiles. Despite these advancements, these approaches mainly focus on volumetric video quality alone rather than QoE. Addressing this gap, Wang et al. [105] introduce a novel QoE metric specifically designed for tile-based volumetric video streaming. They formulate the QoE optimization problem as a multiple-choice knapsack problem (MCKP). Li et al. [60] further consider the computation overhead brought by volumetric video coding. They explore the balance between bandwidth and computing resources across different tile compression versions and qualities. Similarly, Li et al. [59] investigate the balance between bandwidth and computing resources with a new hybrid saliency-based 3D tiling scheme. Recent work extends tile-based strategies to neural scene representations. LTS [94] combines spatial tiling, temporal segmentation, and multi-layer encoding in a DASH-based pipeline to efficiently stream dynamic 3D Gaussian Splatting scenes. FSVFG [117], though not using explicit tiles, achieves similar adaptation through feature grids and selective transmission of scene regions, making it a practical alternative for large-scale dynamic content.

Despite the significant advancements in tile-based volumetric video streaming, current approaches predominantly assume that the visibility information for each tile is known in advance. In reality, this information is typically obtained through culling techniques and viewport prediction, which brings extra computation overhead. Furthermore, the size of tiles plays a crucial role in the efficiency of streaming, which remains unexplored.

RL-based volumetric videos streaming. The use of Reinforcement Learning (RL)

in volumetric video streaming effectively addresses challenges related to uncertain future information, such as dynamic viewports and fluctuating bandwidth conditions. Trans-RL [124] leverages the transformers for viewport prediction and employs DQN for single-user QoE optimization. Expanding on this, Lin et al. [64] explore multi-user scenarios, focusing on long-term QoE enhancement. The POT framework [57] specifically addresses the challenge of long-term prediction errors by adopting a rolling optimization strategy using Serial Cyclic Dueling DQN (SC-DDQN). This strategy focuses on continuous prediction and optimization of streaming parameters, enhancing responsiveness to changes in user behavior and network conditions. To further improve training performance and protect user data privacy, FRAS [24] proposes the first federated reinforcement learning framework.

While these RL-based methods significantly improve performance handling uncertain future information, they have not yet been fully applied to solve specific challenges in tile-based volumetric video streaming, such as optimizing tile size in real time based on user interaction and network status.

CHAPTER 3

REAL-TIME VIDEO ANALYTICS WITH FLEXIBLE COMPUTING RESOURCE CONTROL

3.1 Introduction

Real-time video analytics requires a large amount of computational resources to process video streams in real time. Existing systems often rely on static CPU-GPU allocation, which fails to adapt to dynamic workloads. The imbalance between CPU-GPU workloads and available resources causes a significant accuracy drop and results in inefficient resource utilization. For example, a 10% – 90% CPU-GPU system (details in Section 3.4) performs well on the 10 frames per second (fps) video, but is not effective on the 30 fps video, because the 30 fps video has more CPU workloads and fewer GPU workloads than the 10 fps video.

Inspired by Gemini [36], we leverage the newly developed hardware, dual-image FPGAs, to enhance video analytics by providing flexible CPU-GPU resources. Dual-image FPGAs support runtime switching between two pre-configured FPGA images. This allows

Table 3.1: Comparison with the existing work

	Objective	Multi Users	CPU Functions	GPU Functions	Resource Adjustment
Deepdecision [115]	Composite utility of frame rate and accuracy	No	Resizing	Multiple DNN/CNN models	GPU-dominant
FastVA [101]	Accuracy; Composite utility of delay and accuracy	No	Resizing	Multiple DNN/CNN models	GPU-dominant
JCAB [104]	Composite utility of energy consuming and accuracy	Yes	Resizing	Multiple DNN/CNN models	GPU-dominant
O^3 [29]	Accuracy	No	Filtering, tracking	Single DNN/CNN model	Fixed CPU-GPU
MARLIN [10]	Balance between energy and accuracy	No	Change detector, tracking	Single DNN/CNN model	Fixed CPU-GPU
AdaVP [67]	Accuracy	No	Resizing, Tracking	Single DNN/CNN model	Fixed CPU-GPU
Gemini [36]	Accuracy		Filtering, background subtraction, cropping	Multiple DNN/CNN models	Flexible CPU-GPU
Ours	Accuracy	Yes	Tracking	Multiple DNN/CNN models	Flexible

a flexible CPU-GPU framework by pre-configuring the FPGA with a CPU-image for instruction-intensive tasks and a GPU-image for data-intensive tasks. These two images are multiplexed in the time domain, ensuring efficient utilization of computing resources by dynamically switching between different processing patterns as needed.

Limitations of Status Quo. Computing resource control and video analytics con-

figuration selection are two essential components in video analytics systems with flexible CPU-GPU resources. Computing resource control involves dynamic allocation of resources between the CPU and GPU, while video analytics configuration selection includes selecting configurations, such as video solution, sampled frame rate, and the specific DNN/CNN model. Most existing works focus only on video analytics configuration selection in fixed CPU-GPU resource systems, as listed in Table 3.1. GPU-dominant systems like DeepDecision [115] and FastVA [101] optimize resolution and model selection, whereas fixed CPU-GPU systems leverage CPU computing resources. For instance, MARLIN [10] and AdAVP [67] consider object detection (GPU-based) and tracking (CPU-based) by selecting the most suitable DNN/CNN model and tracking parameters. Authors in the work [29] propose a threshold control algorithm for the CPU filtering function using a fixed frame resolution and a DNN/CNN model. These approaches fail to fully exploit the heterogeneous computing resources and adapt CPU-GPU resources to dynamic video contents due to fixed CPU-GPU resources.

Though Gemini [36] addresses some above issues by offering flexible CPU-GPU computing resources, it still struggles with various video scenarios due to an ineffective CPU-GPU workload partition. This inefficiency arises for two main reasons. First, the frame filtering and background subtraction module implemented on the CPU-image do not always effectively reduce the GPU workloads. For example, in a busy traffic scenario, the CPU fails to filter frames, leading to excessive GPU load. Second, Gemini uses a bandit-based approach to address computing resource control and video analytics configuration selection. While bandit algorithms explore different video analytics configurations, they suffer from performance instability in dynamic environments and slow adaptation to sudden video content changes. Additionally, bandit algorithms may not react quickly enough to sudden changes in video content. These limitations hinder Gemini's effectiveness in handling diverse and rapidly changing video scenarios. Moreover, Gemini can

not support multiple users.

Challenges and Contributions. In this chapter, we address these limitations by proposing Gemini plus (Gemini+), a dual-image FPGA-based video analytics pipeline with a more efficient and robust CPU-GPU workload partition strategy and support multiple users. Specifically, we reorganize the FPGA implementation by deploying resizing and tracking modules on the CPU-image to leverage its capability for instruction-driven tasks. In contrast, the GPU-image is configured to execute data-intensive tasks such as DNN/CNN model inference. The tracking module allows us to more efficiently allocate CPU-GPU workloads by adjusting tracking frame proportions. Moreover, we provide a comprehensive model of a video analytics pipeline with dual computing resources and propose novel algorithms for resource allocation and video analytics configuration selection. These algorithms dynamically adjust to the varying workloads and video content, hence improving analytics performance.

First, the optimal CPU-GPU workload partition changes with video characteristics that require the system to adapt quickly to dynamic resource demands. As we mentioned before, videos with different frame rates have different CPU-GPU workload optimizations. Meanwhile, video content leads to dynamic CPU-GPU workloads as well. For example, compared with videos with fast-speed targets, videos with slow-speed targets lead to higher CPU workload, because the tracking module in CPU-image keeps higher accuracy. The optimal CPU-GPU partition strategy and resource control scheme should adapt to diverse videos.

Second, in comparison to fixed computing resource systems, it is challenging to jointly decide the CPU-GPU resource control and configuration selections in the flexible computing resource video analytics system, because they have mutual influences on each other. For example, if we allocate limited computing resources to the GPU-image, it may

not be possible to run a complex DNN model. Conversely, if we increase the number of frames fed to the CPU-image, we need to assign more resources to it. As a result, it is necessary to consider CPU-GPU resource division and configuration selection in a joint manner.

To overcome the above challenges, we need a new problem formulation of the video analytics pipeline with CPU-GPU resource control and configuration selection. We formulate the dual computing resource control problem under two scenarios. In the single-camera scenario, the pipeline runs on FPGA-powered smart cameras, each with flexible CPU-GPU resources. We propose an optimal solution that effectively utilizes the available flexible resources. In the multi-camera scenario, the dual-image FPGA acts as an edge server that provides video analytics services for multiple cameras. We propose a suboptimal yet highly effective solution.

In summary, our contributions are as follows:

- We introduce flexible CPU-GPU resources in a video analytics pipeline by leveraging the dual-image FPGA. The CPU-image contains resizing and tracking functions, while the GPU-image holds several DNN models.
- We formulate the dual computing resource control (DCRC) problem to maximize video analytics accuracy in both the single-camera system and multi-camera scenario. We analyze the problem complexity and propose algorithms for the above two problems.
- We evaluate the performance of our framework through practical simulations. We compare our work with the fixed CPU-GPU resources and GPU-dominant resources methods. Results indicate that the flexible CPU-GPU resources have better performance.

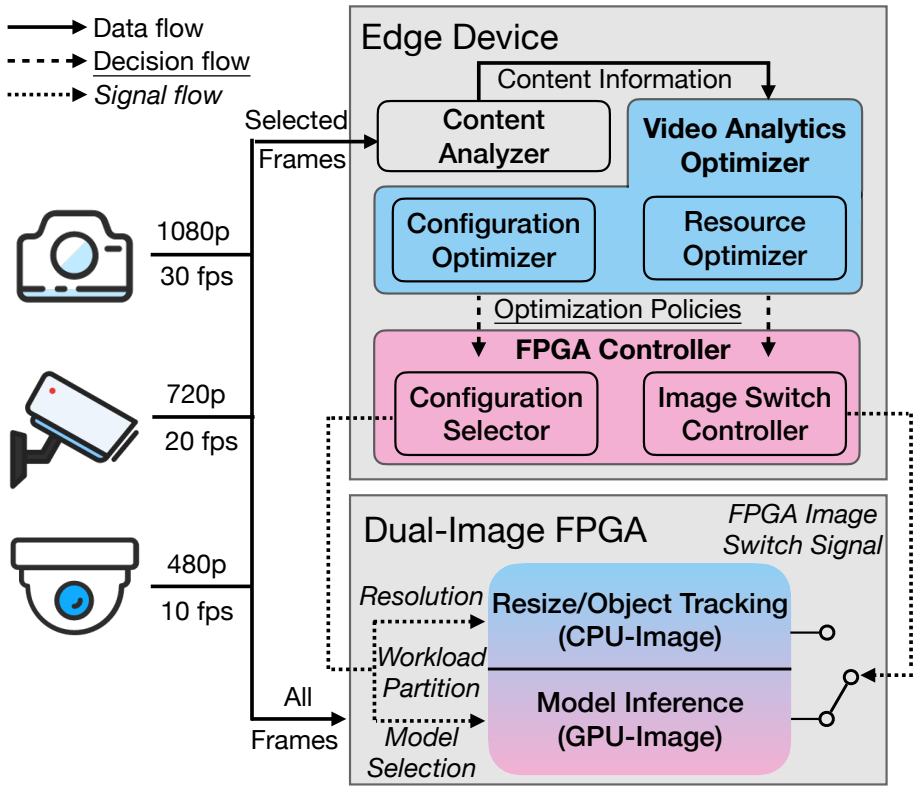


Figure 3.1: System architecture of Gemini+.

3.2 System Model and Problem Formulation of Gemini+

In this section, we first describe the overview of the Gemini+ system architecture. Then we provide detailed models for video analytics optimization in Gemini+, including the video analytics process model, video analytics accuracy model, and system delay model. With these models, we formulate the dual computing resource control problems for the single-camera scenario and the multi-camera scenario.

3.2.1 System Architecture Overview

The architecture of our proposed dual-image FPGA-based system, Gemini+, is designed to provide flexible heterogeneous computing resources for real-time video analytics.

As shown in Fig. 3.1, the Gemini+ system consists of multiple cameras for video capture, an edge device[1] responsible for video analytics optimization and FPGA control, and a dual-image FPGA that performs specific video analytics tasks.

Considering that model performance is influenced by video content, the **content analyzer** in the edge device periodically selects frames to extract critical information, such as object size and motion speed, enabling the system to adapt dynamically to diverse video scenarios. We select object size and speed as video features because they significantly influence DNN/CNN model performance. Additionally, these features can be efficiently extracted with lightweight algorithms, making them well-suited for real-time video analytics. The framework remains flexible and can incorporate additional features if needed. The information is then utilized by the **video analytics optimizer** to generate video analytics optimization policies. These policies guide the **FPGA controller** in computing resource control and video analytics configuration selection, including workload partition, resolution adjustment, and DNN/CNN model selection.

The dual-image FPGA executes video analytics tasks based on signals from the **FPGA controller**. It contains two pre-configured images, a CPU-image and a GPU-image. The CPU-image is optimized for instruction-intensive tasks like resizing and object tracking, while the GPU-image is dedicated to data-intensive tasks like DNN/CNN model inference. The dual-image FPGA runs each image with video analytics configuration and switches the two images according to the image switch signal. An alternative approach is space-division multiplexing, where computing resources are statically partitioned for instruction-intensive and data-intensive tasks on the same image. However, this approach lacks runtime adaptability, as adjusting the resource allocation requires full FPGA reconfiguration, which takes minutes. Additionally, video analytics does not

[1]The edge device can either be a standalone processing unit or a microprocessor embedded within a smart camera.

Table 3.2: Table of notations

Variable	Description
τ_k^c	The proportion of the CPU-image running time for kth video clip
τ_k^g	The proportion of the GPU-image running time for kth video clip
T^c	The total CPU-image running time
T^g	The total GPU-image running time
T	The time constraint in an epoch
r_k	The frame resolution for kth video clip
$x_{i,k}$	The indicator of the jth DNN model for kth video clip
γ_k^t	The proportion of tracking frames for kth video clip
γ_k^d	The proportion of detection frames for kth video clip
a_k	The accuracy of the kth video clip
D_k^{cpu}	The CPU-image delay for the kth video clip
D_k^{gpu}	The GPU-image delay for the kth video clip
D_k	The total delay for the kth video clip
D	The system delay

always require CPU and GPU operations to run simultaneously, as video frames arrive sequentially. In such cases, space-division multiplexing may lead to resource idleness because one type of processing unit remains underutilized while waiting for the other to complete. In contrast, the time-division design ensures optimal utilization of computing resources by dynamically switching between pre-configured images within 1 ms, making it more suitable for diverse and dynamic video analytics workloads.

In summary, the Gemini+ system aims to utilize heterogeneous computing resources in video analytics by dynamically adjusting the running time of the CPU-image and GPU-image. Meanwhile, it optimizes video analytics configurations in real-time to maximize accuracy. This flexible architecture makes Gemini+ adaptable for various video scenarios, ensuring high performance across diverse conditions. The following subsections present a detailed system model and problem formulation for computing resource control and video analytics configuration selection in Gemini+. The notations used in the modeling are listed in Table 3.2.

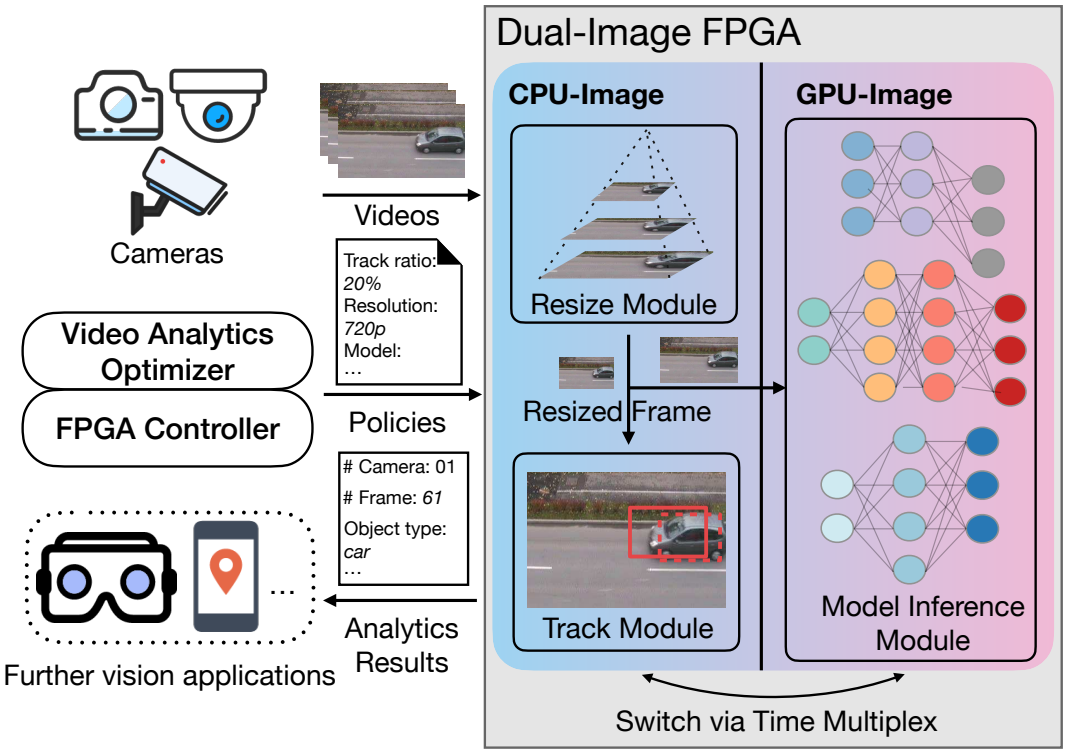


Figure 3.2: Video analytics pipeline of Gemini+.

3.2.2 Modeling the Video Analytics Process

The video analytics pipeline in Gemini+ is composed of three key stages: resizing, object tracking, and DNN/CNN model inference, as illustrated in Fig. 3.2. The process starts with cameras capturing videos, which are then uploaded as video clips at the start of each epoch, with a duration of T. During each epoch, the **FPGA controller** transmits optimization policies generated by **video analytics optimizer**, including video analytics configuration selection and the FPGA image switch scheme. Based on these policies, the video clips are resized to the desired resolution and directed to either the tracking module or the model inference module.

Gemini+ supports video inputs with varying frame rates and resolutions. At each epoch, a video clip v_k captured by the k-th camera has features of a frame rate f_k and a resolution r_k^o. Therefore, the k-th video clip has $n_k = Tf_k$ frames that need to

be processed in an epoch. The resizing step is crucial because high-resolution videos can lead to high accuracy but also result in long processing delays. Therefore, to meet real-time requirements, the input video is resized to a lower resolution. The resizing module adjusts the frame size to the desired resolution r from the resizing resolutions set $\mathcal{R} = \{r_1, r_2, ..., r_R\}$. After resizing, the system should select a DNN/CNN model m from the models set $\mathcal{M} = \{m_1, m_2, ..., m_M\}$ to detect objects in videos. However, it is time-consuming to process all frames via DNN/CNN models, especially when dealing with large volumes of video data. To address this challenge, Gemini+ takes advantage of the spatiotemporal correlations within video frames. Object tracking is used to estimate the position of objects across multiple frames instead of model inference. Consequently, a proportion of frames γ^t are sampled to be processed by the tracking module in the CPU-image, and a proportion of frames γ^d are handled by DNN/CNN model inference in the GPU-image, while other frames $(1 - \gamma^t - \gamma^d)$ are dropped.

As for the FPGA image switch scheme, we adopt a time division multiplexing image switch approach in dual-image FPGAs, where the CPU-image and GPU-image run at separate times. Therefore, the CPU-GPU resource control can be considered equal to the time division. We need to decide on resource division for each video clip τ_k^c, τ_k^g and the total resource division T^c, T^g.

In summary, for each video clip, **video analytics optimizer** need to determine the appropriate resolution r_k, DNN/CNN model m_k, tracking frames proportion γ_k^t and detection frames proportion γ_k^d, collectively known as the video analytics configuration $c_k = (r_k, m_k, \gamma_k^t, \gamma_k^d)$. These decisions are critical to ensuring optimal resource division and system performance.

3.2.3 Modeling the Video Analytics Accuracy

The video analytics system aims to maximize accuracy within time constraints. The analytics accuracy a is measured by the average accuracy of K video clips,

$$a = \frac{1}{K}\sum_{k=1}^{K} a_k. \tag{3.1}$$

The accuracy a_k for the video clip k is jointly determined by the detection accuracy a_k^d and sampling discount rate a_k^s. The detection accuracy a_k^d is based on detection results of DNN/CNN models, while the sampling discount rate a_k^s is determined by the proportion of frames not processed by DNN/CNN models. According to the previous work [104], we assume that the impact of detection accuracy a_k^d and sampling discount rate a_k^s are independent, and the accuracy is the multiplication of detection accuracy and sampling discount rate,

$$a_k = a_k^d a_k^s. \tag{3.2}$$

Detection accuracy. The detection accuracy a_k^d is jointly determined by the DNN/CNN model $m_{i,k}$ and the video resolution r_k. A binary indicator $x_{i,k}$ is used to denote DNN/CNN model selection,

$$x_{i,k} = \begin{cases} 1, & \text{Model } i \text{ is selected for the video clip } k, \\ 0, & \text{Otherwise.} \end{cases}$$

Given the model accuracy function $a_{i,k}^m(r_k)$ of each DNN/CNN model with resolution r_k, the detection accuracy a_k^d is the selected model accuracy,

$$a_k^d = \sum_{i=1}^{M} x_{i,k} a_{i,k}^m(r_k). \tag{3.3}$$

According to the previous work [104] and our experiments results in Section 3.4, the model accuracy function $a_{i,k}^m$ is a concave function with the resolution r_k, $a_{i,k}^m(r_k) = c_{i,k,1}^d - c_{i,k,2}^d e^{-\frac{r_k}{c_{i,k,3}^d}}$, where $c_{i,k,1}^d, c_{i,k,2}^d, c_{i,k,3}^d$ are three fitting coefficients for the concave

function, which can be obtained by experiments. The model accuracy functions vary for different DNN/CNN models and video content. For instance, the resolution has a greater impact on complex DNN/CNN models, such as Yolov3, compared to simpler ones, like Yolov2-tiny. Additionally, the detection accuracy is influenced by the characteristics of the objects in the video. For example, a small object benefits more from a higher resolution than a larger object. Thus, we require different accuracy functions profiling for different video clips and DNN/CNN models.

Sampling discount rate. Due to the limited time, only a proportion of frames γ^d can be processed by DNN/CNN models. Then a proportion of the rest γ^t is proposed by the tracking module, while others $1-\gamma^t-\gamma^d$ are skipped. Both tracking and skipped frames can have a negative effect on detection accuracy. In tracking frames, the position of the detection box obtained from DNN/CNN model inference results, is adjusted based on tracking results. As for the skipped frame, we use the same result as the nearest unskipped frame. Although both methods may result in a reduction of accuracy, tracking frames typically offer superior performance compared to skipped frames, as they utilize more information between frames. We use the sampling discount rate a_k^s to denote such a negative effect. The discount rate is jointly determined by the proportion of detection frames γ^d and tracking frames γ^t. Given the fixed proportion of detection frames γ^d, the sampling discount rate increases with tracking frames γ^t. Conversely, when the proportion of skipped frames $1-\gamma^t-\gamma^d$ is fixed, the sampling discount rate decreases with tracking frames γ^t. We introduce the sampling discount rate function $a_k^s(\gamma_k^d,\gamma_k^t)$, to quantify the impact of detection and tracking frame proportions. However, it lacks a specific mathematical formula due to its complexity and variability. Instead, the function values are derived from experiments.

3.2.4 Modeling the Video Analytics Delay

The system delay is the sum of processing time for each video clip,

$$D = \sum_{k=1}^{K} D_k. \tag{3.4}$$

The k-th video clip delay D_k consists of the CPU processing delay D_k^{cpu} and GPU processing delay D_k^{gpu},

$$D_k = D_k^{cpu} + D_k^{gpu}. \tag{3.5}$$

CPU processing delay. In the CPU-image, both resizing and tracking functions take time. We use an interpolation resizing method and a feature-based tracking method in the CPU-image. The running time of the interpolation resizing method $t^r(r_k)$ is proportional to the square of resolution of the output figure r_k,

$$t^r(r_k) = T_r r_k^2, \tag{3.6}$$

where T_r is the resizing time for each pixel. Exceptionally, if the output resolution is equal to the original resolution, i.e., $r_k = r_k^o$, we regard that the frame skips the resizing method and hence the resizing time is 0 in this situation. Similarly, the tracking delay $t^t(r_k)$ is proportional to the resolution of the input frame,

$$t^t(r_k) = T_t r_k^2, \tag{3.7}$$

where T_t is the tracking time for each pixel. Hence, in the CPU-image, $n_k(\gamma_k^t + \gamma_k^d)$ frames are resized and $n_k\gamma_k^t$ frames are tracked. The CPU processing delay of a single video clip is the sum of all frame delay,

$$D_k^{cpu} = n_k((\gamma_k^t + \gamma_k^d)t^r(r_k) + \gamma_k^t t^t(r_k)). \tag{3.8}$$

GPU processing delay. In the GPU-image, the object detection time of the DNN/CNN model $m_{i,k}$ with the frame resolution r_k is denoted by $t_i^d(r_k)$. With the binary indicator

variable $x_{i,k}$, the GPU-image processing delay can be calculated as

$$D_k^{gpu} = n_k \gamma_k^d \sum_{i=1}^{M} x_{i,k} t_i^d(r_k). \tag{3.9}$$

Similar to the previous works [31, 43], we use the floating-point operations (FLOPs) to measure the detection time,

$$t_i^d(r_k) = F_i r_k + c_i^t, \tag{3.10}$$

where the F_i, c_i^t represent the number of FLOPs required per unit resolution by the selected DNN/CNN model i, and c_i^t denotes the model-specific constant time overhead. All notations are listed in Table 3.2.

3.2.5 Problem Formulation and Complexity Analysis

Based on the above system models, we formulate the dual computing resource control problem in this subsection. The DCRC problem refers to the general optimization problem of jointly selecting video analytics configurations (e.g., resolution, model, frame sampling ratios) and allocating CPU-GPU resources, with the objective of maximizing analytics accuracy under system delay constraints. We first consider the single-camera case, where only one video stream is processed. This leads to the single-camera DCRC (SC-DCRC) problem. We then extend the formulation to the multi-camera DCRC (MC-DCRC) problem, where multiple video clips share computing resources. The MC-DCRC problem is a general form of DCRC, while the SC-DCRC can be viewed as a special case of MC-DCRC with $K = 1$.

3.2.5.1 The Single-Camera DCRC Problem

We first consider the DCRC problem when there is only one camera in the system. The case is valuable when the FPGA-assisted smart cameras are available as works [71, 106] do. This problem provides insights into the optimal video analytics configuration selection and the CPU-GPU resource control.

The single-camera problem indicates $K = 1$ and therefore we drop the user index k in this scenario for brevity. In this case, we do not need to consider the time allocation among multiple cameras, but we should determine the optimal video analytics configuration selection and CPU-GPU resource control. We formulate the single-camera dual computing resource control problem as follows,

$$P1: \max_{\mathbf{x},r,\gamma^c,\gamma^d,T^c,T^g} a(\mathbf{x},r,\gamma^c,\gamma^d) \tag{3.11}$$

$$\text{s.t. } D^{cpu} \leq T^c \tag{3.12a}$$

$$D^{gpu} \leq T^g \tag{3.12b}$$

$$T^c + T^g \leq T \tag{3.12c}$$

$$\sum_{i=1}^{M} x_i = 1, x_i \in \{0,1\}. \tag{3.12d}$$

Theorem 3.1. *The SC-DCRC problem can be solved in polynomial time.*

Proof. We can design an algorithm to search the solution space. We need to search all possible values for each parameters, i.e., the DNN model in $\mathcal{M}$, resolution in R, tracking frames and detection frames $\{\frac{1}{n}, \frac{2}{n}, ..., 1\}$, CPU-image and GPU-image resource in $\{\frac{1}{n_T}, \frac{2}{n_T}, ..., 1\}$. As such, the total solution space is $O(M \times R \times n^2 \times n_T^2)$. ■

Though the problem is polynomial, a brute force search of the solution space is still time-consuming. We will develop a much more efficient algorithm in Section 3.3.

3.2.5.2 The Multi-Camera DCRC Problem

In the multi-camera system, the FPGA acts as an edge server that provides video analytics services for multiple cameras as [61, 109] do. In this system, we should allocate

CPU-GPU resources and select the optimal configuration for each video clip. We formulate the multi-camera dual computing resource control problem as follows,

$$P2: \max_{\mathbf{x},\mathbf{r},\vec{\gamma}^d,\vec{\gamma}^t,\vec{\tau}^c,\vec{\tau}^g,T^c,T^g} \frac{1}{K}\sum_{k=1}^{K} a_k(x_k, r_k, \gamma_k^c, \gamma_k^d) \tag{3.13}$$

$$\text{s.t. } D \le T \tag{3.14a}$$

$$D_k^{cpu} \le \tau_k^c T^c \tag{3.14b}$$

$$D_k^{gpu} \le \tau_k^g T^g \tag{3.14c}$$

$$T^c + T^g \le T \tag{3.14d}$$

$$\sum_k \tau_k^c = 1, \sum_k \tau_k^g = 1 \tag{3.14e}$$

$$\sum_{i=1}^{M} x_{i,k} = 1, x_{i,k} \in \{0,1\}, \forall v_k \in V. \tag{3.14f}$$

The objective function (3.13) aims to maximize the average accuracy in an epoch. The problem follows the constraints:

- The system delay should not exceed the time constraint (3.14a).
- The CPU-Image delay of a single video clip should not exceed the assigned CPU-image time(3.14b).
- The GPU-Image delay of a single video clip should not exceed the assigned GPU-image time(3.14c).
- The total CPU-Image and GPU-Image running time should be less than the time constraint (3.14d).
- All cameras share the CPU-Image and GPU-Image processing time (3.14e).

- One and only one DNN model can be selected (3.14f).

Theorem 3.2. *The MC-DCRC problem is NP-hard.*

Proof. We prove this theorem by transforming the problem into the 2-D bounded knapsack problem (BKP). Consider a special case that video clips are processed by the GPU-image only, i.e., $r_k = r_{ori}, \gamma_k^t = 0, \tau_k^c = 0, \forall k \in \{1,2,...,K\}$. In this way, only constraint (3.14a) and (3.14f) remain. Let DNN models be the items and hence the MC-DCRC problem is equivalent to the 2-D BKP, which aims to maximize the value by determining the number of each item $\mathbf{x} = \{x_1, x_2, ..., x_n\}$ within two constraints. ■

3.3 Dual Computing Resource Control Algorithms

In this section, we analyze and propose algorithms for the SC-DCRC problem and the MC-DCRC problem.

3.3.1 The Single-Camera DCRC Algorithm

Though we could find a polynomial-time brute force algorithm as Theorem 3.1 indicates, it is impractical especially when there is a large number of candidate parameters to select. We need to reduce the algorithm complexity by leveraging the mathematical structure of the problem.

The search space can be divided into two parts, i.e., the video analytics configuration selection and CPU-GPU resource control. In the video analytics configuration selection, Theorem 3.3 helps reduce the search space of resolution by providing a resolution threshold. Theorem 3.4 provides the optimal relationship between detection and tracking frames. In the CPU-GPU resource control part, Theorem 3.5 gives the best resource division strategy. With these theorems, we can reduce the search space significantly.

Definition 3.1. The detection configuration $c_1^d = (m_{i_1}, r_{i_1})$ is said to be **superior** to the detection configuration $c_2^d = (m_{i_2}, r_{i_2})$, if $a^*(c_1^d) > a^*(c_2^d)$ where $a^*(c^d)$ is the optimal value of the objective function (3.11) with the corresponding detection configuration.

Lemma 3.1. *The detection configuration $c_1^d = (m_{i_1}, r_{i_1})$ is superior to the detection configuration $c_2^d = (m_{i_2}, r_{i_2})$, if the detection accuracy $a^d(c_1^d) \geq a^d(c_2^d)$ and the accuracy per latency $\frac{a^d(c_1^d)}{t_1^d} > \frac{a^d(c_2^d)}{t_2^d}$.*

Proof. For simplification, we use a_1^d, a_2^d to represent the detection accuracy $a^d(c_1^d), a^d(c_2^d)$ in the proof. We assume that the relationship between two detection configurations is $a_1^d = k_a a_2^d$ and $t_1^d = k_t t_2^d$ where $k_a, k_t > 1$ and $k_a > k_t$. Let γ_2^d, γ_2^t be the best proportion of detection and tracking frames with the detection configuration c_2^d. We have the optimal accuracy as

$$a_2^* = a_2(\gamma_2^d, \gamma_2^t) = a_2^d a^s(\gamma_2^d, \gamma_2^t) \tag{3.15}$$

We choose $\frac{\gamma_2^d}{k_a}, \gamma_2^t$ for the detection configuration c_1^d which satisfies the total time constraint Eq. (3.12c) because

$$D_1(\frac{\gamma_2^d}{k_a}, \gamma_2^t) < D_1(\frac{\gamma_2^d}{k_t}, \gamma_2^t) \leq D_2(\gamma_2^d, \gamma_2^t) \leq T.$$

Then, we have the accuracy as

$$a_1(\frac{\gamma_2^d}{k_a}, \gamma_2^t) = k_a a_2^d a^s(\frac{\gamma_2^d}{k_a}, \gamma_2^t) = a_2^d k_a a^s(\frac{\gamma_2^d}{k_a}, \gamma_2^t) \tag{3.16}$$

As we mentioned in section 3.2.3, the sampling accuracy function is a concave function with both γ_d, γ_k and hence we have $k_a a^s(\frac{\gamma_2^d}{k_a}, \gamma_2^t) > a^s(\gamma_2^d, \gamma_2^t)$. Because $a_1^* > a_1(\frac{\gamma_2^d}{k_a}, \gamma_2^t)$ and $a_1(\frac{\gamma_2^d}{k_a}, \gamma_2^t) > a_2^*$, we have $a_1^* > a_2^*$. ■

Based on the above definition and lemma, Theorem 3.3 provides a smaller search space of resolution.

Theorem 3.3. *There exists a resolution threshold.*

$$r_i^{th} = f^{-1}(c_{i,3}^d c_{i,1}^d F_i) \tag{3.17}$$

where $f(r) = c_{i,2}^d e^{-r/c_{i,3}^d}(F_i r + c_i^t + c_{i,3}^d F_i)$*, for each DNN/CNN model* i*. With the same DNN/CNN model, the detection configuration with the resolution threshold is superior to the ones with lower resolutions.*

Proof. For the same DNN/CNN model, the detection accuracy increases with the resolution, i.e., $a^d(i, r_i^{th}) > a^d(i, r_i), r_i^{th} > r_i$. According to the Lemma 3.1, we are able to find the resolution threshold r_i^{th} by solving the following problem,

$$P1.1 : \max_{r_i^{th}} \frac{a^d(i, r_i^{th})}{t_i^d(r_i^{th})}. \tag{3.18}$$

The resolution threshold is $r_i^{th} = f^{-1}(c_{i,3}^d c_{i,1}^d F_i)$. ■

Lemma 3.2. *The optimal detection-tracking frame assignment* (γ^d, γ^t) *satisfies* $D^{cpu} + D^{gpu} = T$.

Proof. We assume that the optimal frame assignment solution (γ^d, γ^t) does not satisfy $D^{cpu} + D^{gpu} = T$, we are able to find a new solution $(\gamma^d + \epsilon, \gamma^t)$ satisfies the equation. Because the objective function increases with γ^d, the new solution has higher accuracy than the original one. ■

Theorem 3.4 indicates the optimal detection-tracking frame assignment so that we do not need to try all detection and tracking frames combinations.

Theorem 3.4. *The optimal detection-tracking frame assignment* (γ^d, γ^t) *satisfies*

$$\gamma^t = \frac{\frac{T}{f} - \gamma^d(t^d + t^r)}{t^t + t^r}. \tag{3.19}$$

Proof. According to the Lemma 3.2, we have

$$f\gamma^t t^t + f(\gamma^t + \gamma^d)t^r + f\gamma^d t^d = T.$$

Thus, we can get the relationship between the proportion of detection frames and tracking frames

$$\gamma^t = \frac{\frac{T}{f} - \gamma^d(t^d + t^r)}{t^t + t^r}.$$

■

Definition 3.2. The direct resource division T^{c*}, T^{g*} is a feasible resource division for the configuration $c = (m, r, \gamma^d, \gamma^t)$ which satisfies $D^{gpu}(m, r, \gamma^d) = T^{g*}$. Other feasible resource division schemes are said to be indirect resource divisions.

Theorem 3.5 describes that resource divisions do not affect accuracy as long as they are feasible for the same configuration. Therefore, we only need direct resource division instead of searching the whole resource division space.

Theorem 3.5. *Direct resource division has the same accuracy as indirect resource divisions for the same configuration.*

Proof. We prove it by contradiction. We assume the accuracy for the configuration c with the direct resource division is a^{di} and the accuracy with the indirect resource division is a^{in} and $a^{di} \neq a^{in}$. According to the objective function (3.11), we have $a^{di} = a(c)$ and $a^{in} = a(c)$ which indicates that $a^{di} = a^{in}$ and it is contradict to the assumption. ■

Based on the above theorems, we propose an efficient SC-DCRC algorithm as Alg. 3.1 shows. First of all, we try each DNN model (line 2). Given the DNN model i, we find the resolution threshold $\bar{r}_i$ based on Theorem 3.3 (line 3). Only the resolutions greater than $\bar{r}_i$ are taken into consideration (line 4). Then according to Theorem 3.4, we are able to

Algorithm 3.1 The SC-DCRC algorithm

Input: $T, \mathcal{M}, \mathcal{R}, a(\cdot)$

Output: $\gamma^{d*}, \gamma^{t*}, r^*, m^*, T^{c*}, T^{g*}$

1 $a^* \leftarrow 0$
2 **foreach** *DNN model* $i \in \mathcal{M}$ **do**
3 Obtain the resolution threshold $r_i^{th} = f^{-1}(c_{i,3}^d c_{i,1}^d F_i)$ by Theorem 3.3
4 **foreach** *Resolution* $r \in \mathcal{R}, r \geq r_i^{th}$ **do**
5 $\mathrm{n}_d^{max} \leftarrow \min\{n, \lfloor \frac{T}{(t^s(r)+t^r(r))} \rfloor\}$
6 **for** *Detection frames* $n_d \leftarrow 1$ **to** n_d^{max} **do**
7 $\gamma^d = \frac{n_d}{f}$
8 Obtain the proportion of tracking frames $\gamma^t = \frac{\frac{T}{f} - \gamma^d(t^d + t^r)}{t^t + t^r}$ by Theorem 3.4
9 **if** $\gamma^t \geq 0$ *and* $a(i, r, \gamma^t, \gamma^d) > a^*$ **then**
10 $a^* \leftarrow a(i, r, \gamma^t, \gamma^d)$
11 $m^* \leftarrow i, r^* \leftarrow r$
12 $\gamma^{d*} \leftarrow \gamma^d, \gamma^{t*} \leftarrow \gamma^t$
13 **end**
14 **end**
15 **end**
16 **end**
17 Obtain the direct resource division (T^{g*}, T^{c*})
18 **return** $\gamma^{d*}, \gamma^{t*}, r^*, m^*, 1 - T^{g*}, T^{g*}$

find the best proportion of detection and tracking frames γ^d, γ^t (lines 5-8). We keep the configuration with the highest accuracy (lines 9-12). Finally, a direct CPU-GPU resource division for the optimal configuration by Theorem 3.5 is provided (line 17).

The computation complexity of the SC-DCRC algorithm is $O(|\mathcal{M}| \times |\mathcal{R}| \times n)$ as it searches the space of the DNN/CNN models, partial resolutions and detection frames. Thanks to the polynomial growth in complexity, it also demonstrates good scalability. The configuration search is guided and bounded by theoretical results, including the resolution threshold (Theorem 3.3) and optimal detection-tracking frame assignment (Theorem 3.4), enabling systematic exploration of the relevant configuration space without exhaustively evaluating all combinations. This ensures that the algorithm remains efficient and practical as the configuration space expands.

3.3.2 The Multi-Camera DCRC Algorithm

We propose an efficient heuristic algorithm for the MC-DCRC problem based on several observations.

Observation 3.1. *The MC-DCRC problem can be regarded as a two-step problem. The first step is to allocate time resources for each user and the second step is to find the optimal configurations for each camera given the time resource.*

Given the time allocation, the optimal accuracy $a_k^*(t_k)$ can be derived from the SC-DCRC algorithm 3.1. Therefore, we only need to focus on the time allocation problem in the first step which reduces the original problem as follows,

$$P2.1: \max_{\mathbf{t},\vec{\tau}^c,\vec{\tau}^g,T^c,T^g} \frac{1}{K}\sum_{k=1}^{K} a_k^*(t_k) \tag{3.20}$$

$$\text{s.t. } \tau_k^c T^c + \tau_k^g T^g \le t_k \tag{3.21a}$$

$$T^c + T^g \le T \tag{3.21b}$$

$$\sum_k \tau_k^c = 1, \sum_k \tau_k^g = 1. \tag{3.21c}$$

Theorem 3.6. *The reduced time allocation problem is NP-hard.*

Proof. The reduced time allocation problem can be transformed to the generalized knapsack problem [95] (GKP). Each camera is the item and the value function is the accuracy function with the allocated time $a^*(t)$. We can reduce the 0-1 knapsack problem to the GKP and the 0-1 knapsack problem is NP-hard. Therefore, the GKP is NP-hard and the reduced time allocation problem is NP-hard. ■

Though the reduced problem is still NP-hard, compared with the original problem, the property of the accuracy function $a_k^*(t_k)$ provides us with good insights into the solution.

Observation 3.2. *The optimal accuracy $a_k^*(t_k)$ is a non-decreasing function with the allocated time t_k and it is not always smooth.*

The optimal accuracy function $a_k^*(t_k)$ is a piecewise function whose value is the maximum of each configuration accuracy given the assigned time. Intuitively, this function is non-decreasing with respect to time, as more time can only improve or maintain the same level of performance as seen in previous attempts. With the increase of allocated time, when the DNN model and resolution remain constant, the accuracy function $a_k^*(t_k)$ varies smoothly with changes in the detection and tracking frames γ_k^d and γ_k^t. However, when discrete changes are made to the DNN model or resolution, the accuracy function becomes non-smooth. Then, cusp points appear in the curve due to the sudden shifts in parameters.

Observation 3.3. *When the allocated time is short, the accuracy function $a_k^*(t_k)$ increases rapidly with time. When the allocated time is long, the accuracy function $a_k^*(t_k)$ improvement becomes slow and reaches the plateau frequently.*

When the allocated time is limited, only a portion of frames can be processed, resulting in a rapid improvement in accuracy as more frames are processed. However, when the time becomes long, almost all frames are processed. To continue enhancing accuracy, it becomes necessary to allocate additional time towards selecting more complex DNN models and/or higher resolutions. However, these adjustments lead to a slower rate of improvement, and frequent plateaus. Therefore, we should assign more time to those who have fast accuracy increment tendency while decreasing time for those who have a slow accuracy increase trend. However, it is hard to tell the time allocation threshold for different users directly. We use the slope $s_k = \frac{a_k^*(t_k+\delta)-a_k^*(t_k)}{\delta}$ to measure the sensitiveness to time.

Algorithm 3.2 The MC-DCRC algorithm

Input: $T, \mathcal{M}, \mathcal{R}, a_k(\cdot), \delta$, initial time assignment $\mathbf{t}$
Output: $\mathbf{C} = \{m_k, r_k, \gamma_k^d, \gamma_k^t\}, (T^c, T^g), (\vec{\tau}^c, \vec{\tau}^g)$

1 Obtain $c_k, \vec{\tau}_k^c, \vec{\tau}_k^g$ given t_k by Algorithm 3.1
2 Calculate the objective accuracy a and slop value $\mathbf{s}$
3 **repeat**
4 Select a pair of videos with highest and lowest slope (v_{k1}, v_{k2})
5 $\bar{t}_{k1} \leftarrow t_{k1} + \delta, \bar{t}_{k2} \leftarrow t_{k2} - \delta$
6 $\bar{\mathbf{t}} \leftarrow \{t_1, ..., \bar{t}_{k1}, ..., \bar{t}_{k2}, ..., t_K\}$
7 Obtain new accuracy $\bar{a}$ and results given $\bar{\mathbf{t}}$ by Algorithm 3.1
8 **if** $\bar{a} < a$ **then**
9 $\delta = \alpha\delta$
10 **else**
11 Update the result and keep the new values
12 **end**
13 **until** *The maximum iterations have been reached or the step length exceeds the limitation*;
14 $T^g \leftarrow \sum D_{gpu}(\gamma_k^d, r_k, m_k)$
15 **return** $\mathbf{C}, (1 - T^g, T^g), (\vec{\tau}^c, \vec{\tau}^g)$

The value of δ influences the measurement of the slope. When the δ is extremely small, the value of the slope is 0 mostly because the small change of the allocated time is not able to change the optimal configuration. When the δ is extremely large, we are not able to estimate the tendency around the allocated time. Therefore, we should select a suitable value for it. Besides, we should increase the value of δ when the allocated time reaches the cusp point to overcome the plateau.

Based on the above observations and analysis, we propose the MC-DCRC algorithm as Alg. 3.2 shows. The key idea is to assign more time to the videos with a high improvement rate while allocating less time to those who have a low accuracy increase rate. In Alg. 3.2, an initial solution is achieved by performing the initial time allocation with Alg. 3.1 (lines 1-2). Based on this initial solution, we select a pair of video clips with the highest and lowest slope and re-assign time between them (lines 4-7). To mitigate the risk of getting stuck in local optima, we adaptively adjust the step length. Specifically, if the algorithm fails to improve, we increase the step length by the adjustment factor α (lines

8-12). This adaptive adjustment expands the search space, allowing the algorithm to escape suboptimal solutions and continue converging toward an improved allocation. The adjustment parameter $\alpha > 1$ (line 9) controls the step length growth rate, ensuring a gradual yet effective adaptation. The iterative process terminates when either the maximum number of iterations is reached or the step length exceeds a predefined threshold (line 13), preventing unnecessary computational overhead.

Parameter selection is crucial for the MC-DCRC algorithm under varying video analytics scenarios. When videos are highly diverse, a smaller step length δ and adjustment factor α are preferable to avoid sucking in local optimal. Conversely, when videos are more similar, larger values of δ and α can be employed without a significant accuracy decrease. Moreover, parameter selection is influenced by the delay requirement. Smaller δ and α values result in higher accuracy but slower convergence, making them suitable for applications with longer processing times. In contrast, larger parameters enable faster convergence, making them preferable for latency-sensitive tasks that require real-time processing. The detailed selection of α and δ will be discussed in Section 3.4.

The computation complexity of the proposed MC-DCRC algorithm is determined by its iterative slope-based adjustment process. In each iteration, the algorithm updates the time allocation vector and invokes the SC-DCRC algorithm (Alg. 3.1) for each of the K video streams. Since the SC-DCRC algorithm has a complexity of $O(|M| \times |R| \times n)$, the total complexity of the MC-DCRC algorithm is $O(K \times |M| \times |R| \times n)$. This confirms that the algorithm scales linearly with the number of video streams and polynomially with the configuration space, ensuring its practical tractability and efficiency in real-world multi-camera video analytics systems.

In addition to high computational efficiency, the MC-DCRC algorithm is designed to scale effectively in multi-camera environments. It enables the independent execution

of SC-DCRC for each stream. This structure allows for parallelism and ensures that the overall system complexity grows linearly with the number of cameras. Furthermore, the iterative slope-based adjustment scheme achieves convergence without resorting to exhaustive enumeration of all global combinations, making the approach both scalable and efficient for large-scale deployments.

3.4 Evaluations of Gemini+

In this section, we evaluate the performance of the SC-DCRC and MC-DCRC algorithms to answer the following questions:

- How do configurations affect the accuracy? (Section 3.4.2)
- How is the performance of the SC-DCRC algorithm compared to GPU-dominant and fixed CPU-GPU frameworks? (Section 3.4.3)
- How is the performance of the MC-DCRC algorithm compared to different time allocation schemes? (Section 3.4.4)

3.4.1 Experiment Setup

Dataset and DNN/CNN Models. The input videos come from the VIRAT [72] and Auburn dataset [3]. The original resolution of the video is 1080 p and we offer six more candidate resolutions, i.e., 720 p, 480 p, 360 p, 240 p, 180 p, 90 p. The video frame rate can be set from 2 to 30 fps. We assume that each epoch is one second long. The epoch duration is set to one second, which balances the trade-off between control overhead and system latency. A shorter epoch would require more frequent optimization and increase scheduling cost, while a longer epoch introduces additional delay from video capture to analytics output. Under this setting, the number of frames per video clip ranges from 2

to 30, depending on the frame rate. The value of each epoch can be adjusted in practice according to application-specific latency and efficiency requirements. There are three different DNN/CNN models, i.e., Yolov3, Yolov2, and Yolov2-tiny. They have different accuracy performance and processing time.

Evaluation Metrics. We use the vehicle detection task to evaluate the proposed algorithms. We measure analytics accuracy using the metric of the intersection over union (IOU) [17], similar to Deepdecision [115]. The IOU of a single object is the proportion of the intersection area to the union area of the bounding boxes. The frame IOU is the average IOU of multiple objects in this frame, and the video IOU is the average IOU of multiple frames. The ground truth is given by the Yolov5 model executed on 1080 p raw videos.

Accuracy function profile. As we mentioned before, the accuracy function differs from video clip to video clip. However, it is impractical to profile an accuracy function for each individual video clip due to the extensive time required. To facilitate lightweight profiling, we pre-define several accuracy functions based on the size and moving speed of targets. Then, we select suitable accuracy functions for coming video clips by measuring the size and moving speed of targets in the first two frames. Though pre-defined profiling is not as accurate as individual profiling, it is more time-efficient and feasible for real-time applications.

Processing time profile. The processing time of resizing, tracking, and model inference usually remains stable, and hence, we simply regard the processing time in the current epoch as the same as the time in the previous epoch. Besides, the running time of the SC-DCRC algorithm is less than 1 ms, and the running time of the MC-DCRC algorithm can be controlled by the iteration times. The switching time between CPU-image and GPU-image is around 9 ms. They are negligible compared to the epoch

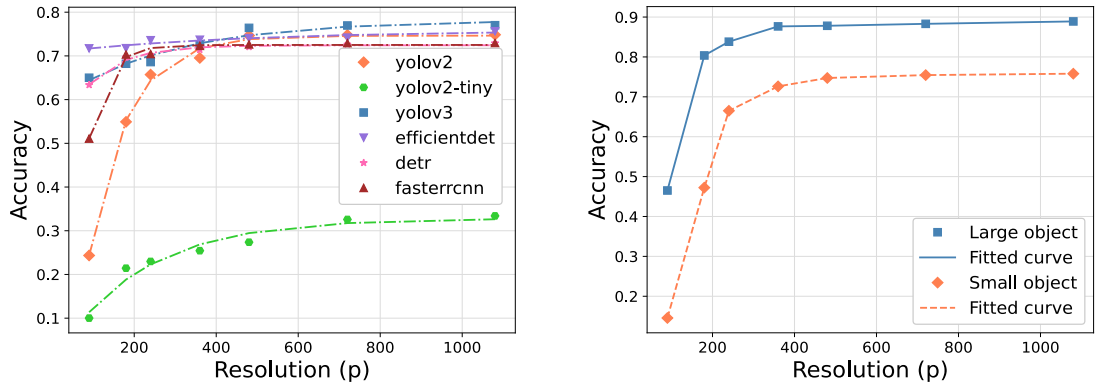
duration, i.e., 1 s.

Baselines. We compare the SC-DCRC algorithm with one state-of-the-art GPU-dominant video analytics framework, one system with fixed CPU-GPU resources, one flexible CPU-GPU system and the offline optimal algorithm.

- **Deepdecision (DD) [115].** It is a GPU-dominant optimization framework. It selects the resolution and DNN/CNN model for each video clip.
- **MARLIN [10].** It is a video analytics pipeline with fixed CPU-GPU resources. We modify the original MARLIN by replacing the change detector module with a tracking frame proportion selection. The CPU and GPU resources are set to a fixed 50%-50% split.
- **Gemini [36].** It is a video analytics pipeline with flexible CPU-GPU resources. It has multiple versions for different video applications. We select the one with a frame filtering module in CPU-image for comparison.
- **Offline optimal (OPT).** This approach is assumed to have foreknowledge of processing outcomes. It simulates running all possible video analytics configurations to identify the one yielding the highest accuracy, serving as a benchmark for the best achievable performance.

We compare the MC-DCRC algorithm with other time allocation schemes.

- **Random time allocation.** The processing time for each video clip is assigned randomly, which is the initial state of the MC-DCRC algorithm.
- **Uniform time allocation.** An equal amount of processing time is allocated to each video clip, disregarding their individual contents and frame rates.



(a) Accuracy vs. models.

(b) Accuracy vs. video contents.

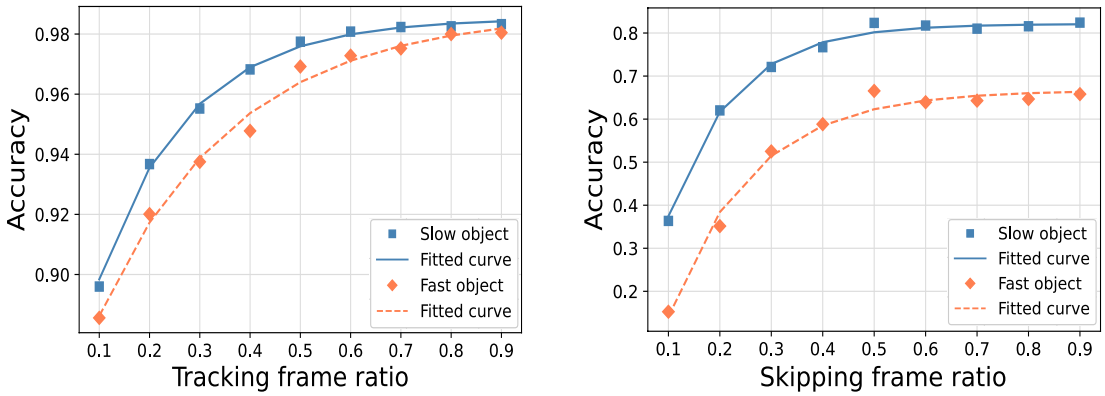
Figure 3.3: Impact of resolution on accuracy.

- **Proportional time allocation.** The processing time is allocated in proportion to the frame rate, aiming to align resource distribution with the total workloads.

3.4.2 The Impact of Configurations on Accuracy

We first conduct experiments with different configurations to understand the impact of these configurations on accuracy.

Impact of resolution and DNN/CNN model. In Fig. 3.3, we plot detection accuracy versus resolution under various DNN/CNN models. It is an increasing function with the resolution no matter which DNN/CNN model is selected, as Fig. 3.3(a) shows. However, the impact of the resolution differs from the DNN/CNN model to the DNN/CNN model. The complex DNN/CNN models, such as Yolov3, DETR [13], and EffcientDet [97], are not as sensitive to changes in resolution as the simple DNN/CNN models, such as Yolov2-tiny, Yolov2, and Faster R-CNN [85]. According to the previous work [104], we have fitted the accuracy function with resolution as a concave function. For example, the blue line can be fitted as $0.782 - 0.189e^{-\frac{r}{0.261}}$. On the other hand, the video content also has an influence on the accuracy function. We show the results in Fig. 3.3(b) where the blue line represents a video with a large object and the orange line indicates a video with a



(a) Accuracy vs. tracking proportion.

(b) Accuracy vs. detection proportion.

Figure 3.4: Impact of detection frames on accuracy.

Table 3.3: Impact of the fixed CPU-GPU resource in optimal configuration selection

CPU-GPU	10 fps Video	20 fps Video	30 fps video
10% - 90%	81.06%	48.60%	32.18%
30% - 70%	**81.90%**	**73.74%**	58.00%
50% - 50%	78.65%	72.62%	**68.35%**
70% - 30%	73.96%	69.28%	67.16%

small object. Even with the same resolution and DNN/CNN model, the large object gains higher accuracy than the small object.

Impact of sampling. We conduct experiments to measure the impact of sampling, as Fig. 3.4 shows. We explore the relationship between the accuracy and the tracking frames in Fig. 3.4(a) given the fixed skipping frames. In Fig. 3.4(b), we investigate the impact of detection frames given the fixed tracking frames. Both relationships can be modeled as a concave function. Meanwhile, video content has an impact on accuracy. For instance, slower objects maintain higher accuracy than faster ones, as the blue and orange lines show.

Impact of CPU-GPU resource control. To measure the impact of CPU-GPU resource control, we select optimal video analytics configurations for three video clips

with different frame rates, i.e., 10 fps, 20 fps, and 30 fps, given different CPU-GPU resource divisions. Table 3.3 shows that lower frame rates generally lead to higher accuracy because the edge server has more processing time per frame. However, the accuracy improvements differ in situations. For example, with 10% CPU and 90% GPU, the accuracy in 10 fps is around 50% higher than the accuracy in 30 fps video clips. But with 70% CPU and 30% GPU, the accuracy in 10 fps is only around 5% higher than the 30 fps video clips.

This variation arises due to distinct characteristics of the CPU and GPU. The CPU can process workloads quickly but with low accuracy, while the GPU has longer processing time but achieves higher analytics accuracy. In the 10%-90% CPU-GPU partition, the GPU has the most of computing resources but it cannot handle excessive workloads. At lower frame rates, the total workload is low, allowing the GPU to handle most of the processing, resulting in higher accuracy. Conversely, at higher frame rates, the GPU cannot manage increased workloads with the same DNN/CNN model. Hence, it switches to simpler DNN/CNN models, leading to lower accuracy. In 70%-30% partition, too many CPU resources decrease accuracy at lower frame rates, but it shares most workloads for the GPU at higher frame rates, minimizing the accuracy drop.

Besides, the optimal CPU-GPU resource allocation varies in different situations. There does not exist a fixed CPU-GPU resources setting that always outperforms others. For example, the 30% CPU and 70% GPU setting has the highest accuracy in 10 and 20 fps video clips, while 50% CPU and 50% GPU has the highest accuracy in 30 fps video clips.

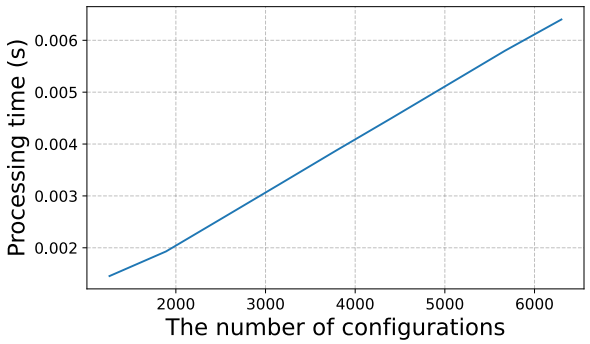


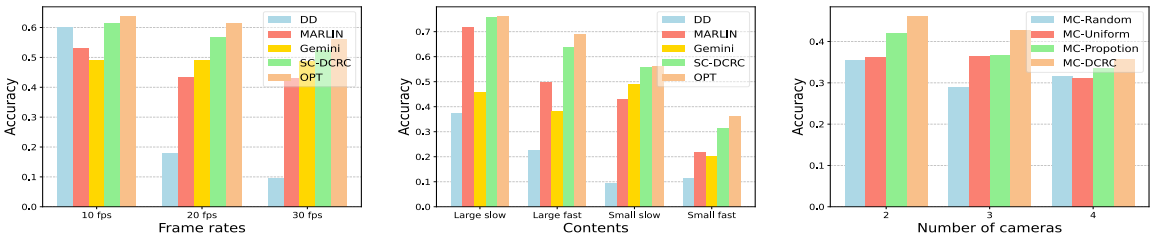
Figure 3.5: Processing time of SC-DCRC with increasing number of configurations.

3.4.3 The Performance of Single-Camera Algorithm

In this subsection, we first analyze the maximum number of configurations the SC-DCRC algorithm can feasibly support. Then we compare the SC-DCRC algorithm to other methods. The performance is highly dependent on the video frame rate and video content.

Maximum number of configurations supported by SC-DCRC. Since the computation complexity of the SC-DCRC algorithm is $O(|\mathcal{M}| \times |\mathcal{R}| \times n)$, its execution time scales linearly with the number of configurations, as shown in Fig. 3.5. The maximum number of configurations that SC-DCRC can process depends on the allocated execution time within each epoch. For instance, if 0.5% of an epoch (i.e., 0.005 s) is allocated to SC-DCRC, the algorithm can support approximately 5,000 configurations while maintaining real-time processing. These results confirm that SC-DCRC remains computationally feasible in real-time scenarios.

The performance of SC-DCRC under different frame rates. In Fig. 3.6(a), we compare the performance of algorithms across different frame rates but similar video contents. Higher frame rates generally lead to lower accuracy due to increased computational workloads. The GPU-dominant method (DD) is particularly affected by the increased workload, with accuracy dropping from 0.6 at 10 fps to 0.2 at 20 fps, and further



(a) Accuracy vs. single-camera frame rates.

(b) Accuracy vs. single-camera video contents.

(c) Accuracy vs. multi-camera user numbers.

Figure 3.6: Performance of different methods in single-camera and multi-camera system.

to 0.1 at 30 fps. Fixed resource systems (MARLIN) perform worse than DD at 10 fps but improve at 20 and 30 fps. This is due to the CPU-image,Äôs tracking module sharing workloads with the GPU-image. Gemini struggles to adjust workload partitioning, leading to similar performance across different frame rates, despite increasing computational demands. SC-DCRC dynamically adjusts CPU-GPU workloads, significantly improving accuracy. It consistently outperforms DD, MARLIN, and Gemini across all frame rates. Even compared to the offline optimal (OPT) method, SC-DCRC achieves comparable accuracy, with a gap of less than 8%, mainly due to a minor difference between profiled and actual accuracy.

The performance of SC-DCRC under different video contents. The performance varies among different video contents, as Fig. 3.6(b) shows. For large, slow-moving objects, all approaches have better performance than with other video contents except for Gemini. This improvement occurs because large objects are easier for DNN/CNN models to detect, and the slower movement facilitates more accurate tracking. However, Gemini performs better with small, slow-moving objects because Gemini tends to feed more frames to the GPU image for large, slow objects compared to small, slow objects, since the frame differences among large moving objects exceed the filtering threshold more frequently. It causes an imbalance in CPU-GPU workloads, leading to lower accuracy. All approaches exhibit the worst performance in videos with small fast objects, except for

Table 3.4: Video features of different cameras

Camera No.	Object size	Object speed	Frame rate
1	Small	Fast	10
2	Large	Fast	30
3	Small	Slow	10
4	Large	Slow	20

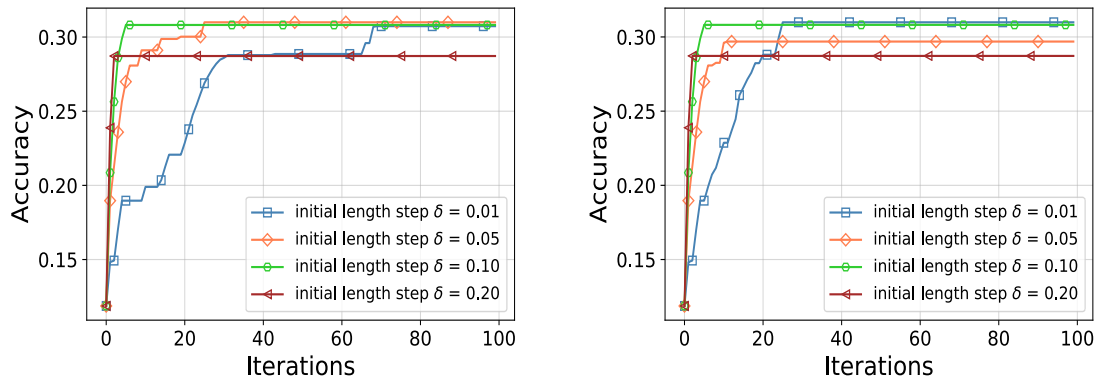
DD. This is because DD relies solely on DNN/CNN models, which are less impacted by object speed than methods that incorporate object tracking modules. In videos with slow objects, SC-DCRC achieves accuracy comparable to OPT because the accuracy function profile is more precise in these situations.

In summary, the experiment results demonstrate the adaptability of SC-DCRC for Gemini+. It performs well in diverse video clips with varying frame rates and contents. Although both Gemini and Gemini+ are video analytics pipelines with flexible computing resources, Gemini+ with SC-DCRC shows superior adaptability to different video contents than Gemini. Besides, the performance gap between SC-DCRC and OPT is small, which indicates that offline profile error is acceptable.

3.4.4 The Performance of Multi-Camera Algorithm

In this subsection, we first compare the performance of different time allocation methods in multi-camera scenarios. Then, we measure the impact of different parameters on MC-DCRC algorithm convergence.

Performance. Fig. 3.6(c) shows the accuracy of different time allocation methods across different numbers of cameras in flexible CPU-GPU resource systems. The video features of these cameras are listed in Table 3.4. The average accuracy decreases as the number of cameras increases due to escalating computation workloads. Among these methods, MC-DCRC always achieves the highest accuracy. Random time allocation ex-



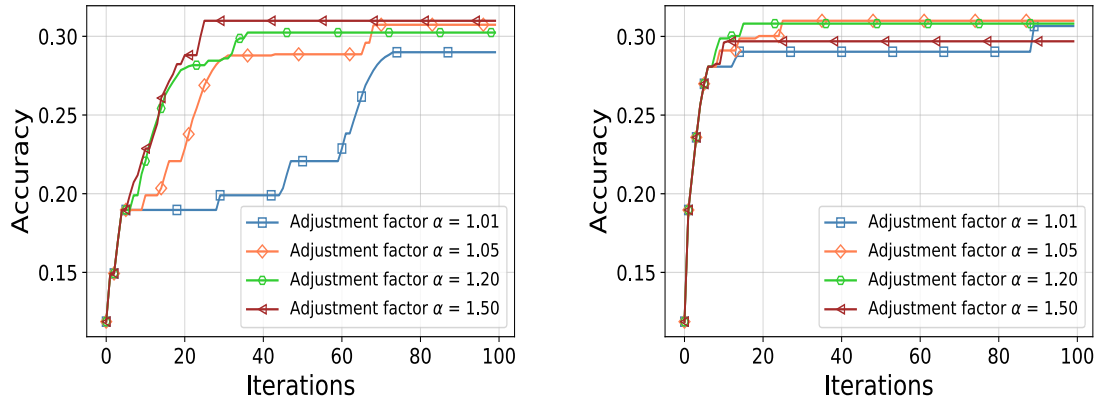
(a) Adjustment factor $\alpha = 1.05$.

(b) Adjustment factor $\alpha = 1.5$.

Figure 3.7: Convergence with initial step length δ.

hibits instability owing to its inherent randomness. MC-DCRC can improve the accuracy by modifying the random time allocation. Uniform time allocation performs poorly when there is significant diversity among video clips. For example, camera 1 and camera 2 have varying video contents and frame rates. Consequently, uniform time allocation exhibits worse accuracy with two cameras compared to three. Proportional time allocation outperforms both random and uniform time allocations because it accounts for the differing computational workloads across cameras. Nevertheless, the optimal time allocation is not only decided by the computation workloads, and hence, it still has worse performance than MC-DCRC.

Convergence. The convergence of the MC-DCRC algorithm depends on two parameters, i.e., the initial step length δ and the adjustment factor α. In Fig. 3.7, we compare the performance of different initial step lengths δ with two adjustment factors, i.e., $\alpha_l = 1.05$ and $\alpha_r = 1.5$. Generally, a larger initial length leads to a faster convergence. When $\delta = 0.01$ and $\alpha = 1.05$, the algorithm converges within around 60 times. But when $\delta = 0.2$, the convergence time decreases to only 3. However, blindly increasing δ would lead to a local optimal and hence achieve low accuracy. In Fig. 3.8, we measure the influence of different adjustment factors α with two initial step lengths $\delta_l = 0.1$ and



(a) Initial step length $\delta = 0.1$.

(b) Initial step length $\delta = 0.2$.

Figure 3.8: Convergence with adjustment factor α.

Table 3.5: Ablation study: accuracy evaluation of SC-DCRC with and without accuracy/delay models

	Large Slow	Large Fast	Small Slow	Small Fast
SC-DCRC	0.76	0.64	0.56	0.32
SC-DCRC-AVG	0.61	0.64	0.54	0.20
SC-DCRC-LAST	0.74	0.64	0.54	0.20

$\delta_r = 0.2$. Similarly, a larger adjustment factor leads to a quick convergence. For example, in Fig. 3.8(a), it takes around 70 times to reach convergence when $\alpha = 1.01$ while it halves the iteration times when $\alpha = 1.5$. However, a larger adjustment factor might get stuck in the inferior solutions, as Fig. 3.8(b) shows. In our experiment, when $\delta = 0.1$ and $\alpha = 1.05$, it achieves a good trade-off between the convergence speed and performance. Currently, we adjust these parameters according to empirical observations.

3.4.5 Ablation Studies

The effectiveness of accuracy and delay models. We compare the SC-DCRC algorithm with two heuristic baselines: 1). **SC-DCRC-AVG**, which replaces the models with precomputed average accuracy and delay values, and 2). **SC-DCRC-LAST**, which uses accuracy and delay from the last epoch, assuming that the video characteristics

remain stable.

Table 3.5 presents a comparative analysis of SC-DCRC and its variants under different video scenarios. Among all methods, SC-DCRC consistently achieves the highest accuracy across all conditions, demonstrating the effectiveness of accuracy and delay models. On average, it improves accuracy by 14% over SC-DCRC-AVG and 7% over SC-DCRC-LAST. The performance gap arises due to the ability of SC-DCRC to dynamically adapt to varying video contents by leveraging real-time accuracy and delay estimations. In contrast, SC-DCRC-AVG assumes a static average accuracy and delay, leading to poor adaptability. The accuracy drops particularly when the video used for profiling differs significantly from the actual videos for analytics. SC-DCRC-LAST partially mitigates this issue by leveraging past analytics results. However, it struggles when video content fluctuates rapidly, as it lacks the capability to handle dramatic changes.

Since the accuracy and delay models are profiled offline, they introduce minimal computational overhead for the SC-DCRC algorithm. The main overhead comes from the content analyzer module, which is responsible for feature extraction. These extracted features are then used for accuracy model selection, ensuring that the most suitable model is applied based on video content. To minimize computational costs, we adopt lightweight algorithms, such as frame difference, which provide a balance between accuracy and processing speed.

3.5 Prototype and Case Study of Gemini+

3.5.1 Prototype Implementation

We implement the Gemini+ prototype system by reorganizing the hardware designs of Gemini. The prototype consists of a Hikvision web camera smart265, an Jiangxing

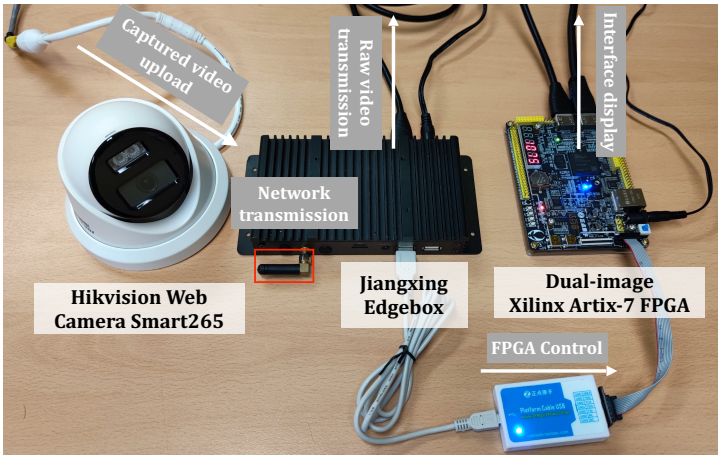


Figure 3.9: System prototype.

edgebox, and a dual-image Xillinx Artix-7 FPGA, as Fig. 3.9 shows. We replace the Intel Max 10 FPGA in Gemini with Xilinx Artix-7 because it provides more processing units to support advanced DNN/CNN models. The camera captures video and uploads it wirelessly to the edgebox for further analysis. The edgebox is responsible for video analytics optimization and FPGA control. Specifically, it receives frames from cameras to perform content analysis. Based on the content information, it generates video analytics optimization policies and FPGA control signals, guiding the FPGA to execute detailed video analytics tasks. After analysis, the FPGA sends the results back to the edgebox.

The edgebox is equipped with a 6-core ARM Cortex-A53 processor, 4GB memory, and runs on Ubuntu 18.04. It receives frames from the camera via wireless transmission and controls the FPGA through a wired connection. The Edgebox transmits raw video data to the FPGA via HDMI for analytics, and the inference results are subsequently transmitted back to the Edgebox through the same HDMI connection, as the current prototype does not include an additional transmission module in the FPGA. The Xilinx Artix-7 FPGA has 101,440 configurable logic cells and 4GB DDR3 memory. The FPGA operates at a maximum clock frequency of 464 MHz. These logic cells are configured into different processing units for the CPU-image and GPU-image, and the DDR3 memory

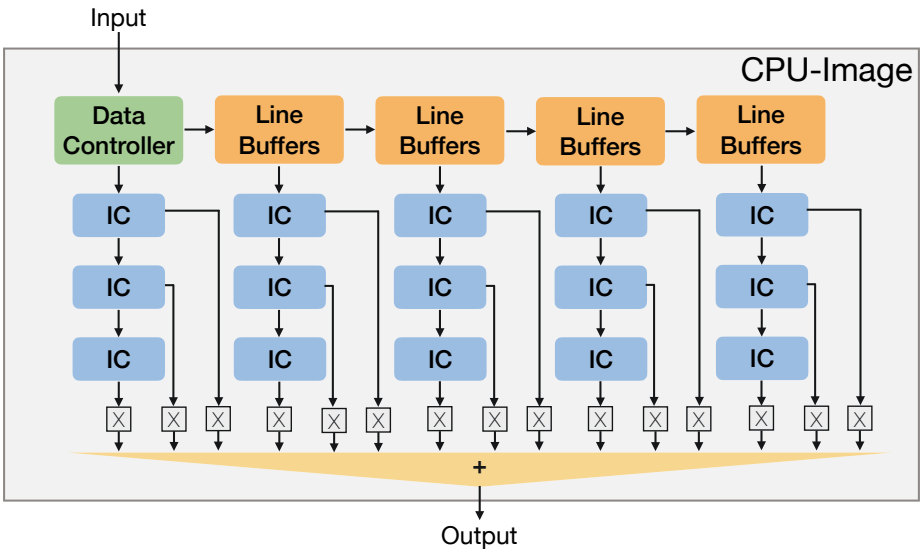


Figure 3.10: A CPU-image implementation.

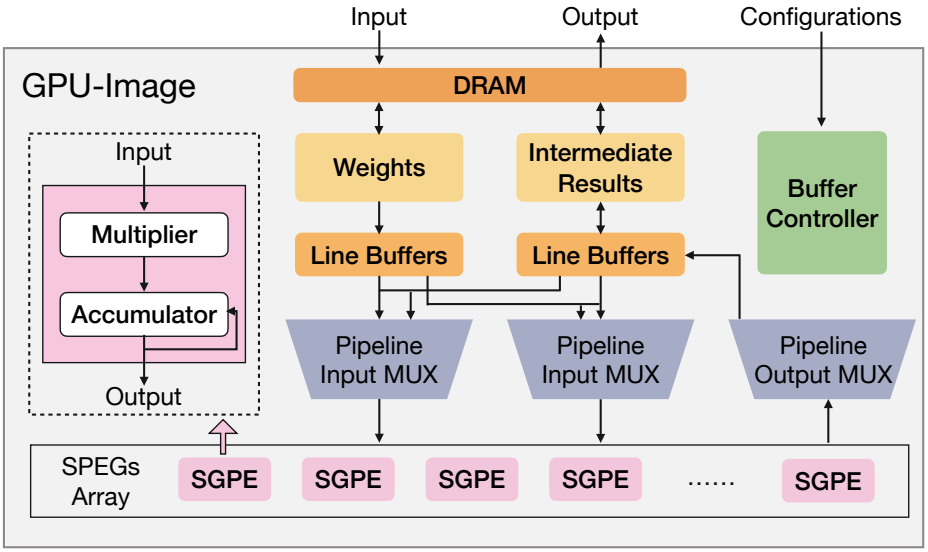
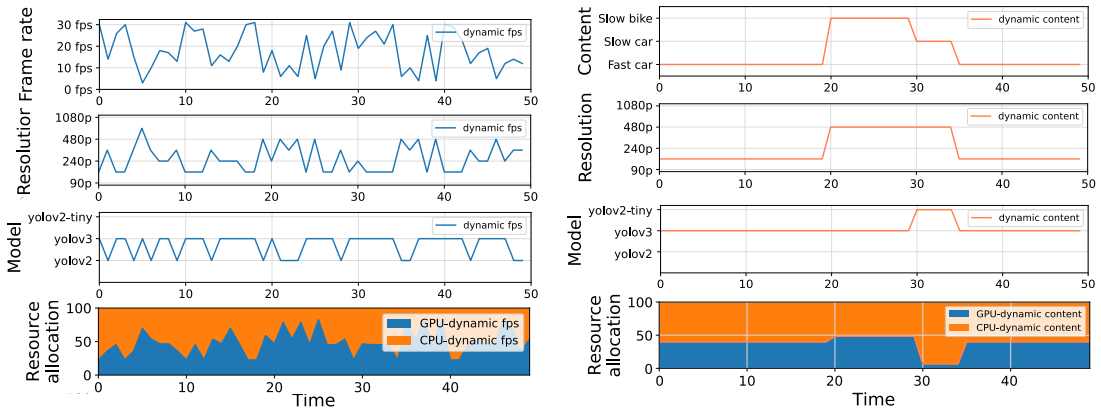


Figure 3.11: A GPU-image implementation.

facilitates data storage and exchange.

Specifically, we implement instruction cores (ICs) as the basic processing unit for the CPU-image, as shown in Fig. 3.10. These ICs calculate the feature values in pixels for bilinear interpolation resizing and optical flow object tracking. To optimize execution efficiency, we adopt a buffer-instruction-core architecture, where data is preloaded into



(a) Dynamic frame rate scenario.

(b) Dynamic content scenario.

Figure 3.12: Runtime video analytics configuration adaptation and resource division of Gemini+.

buffers via the Data Controller. This design ensures that ICs operate continuously without idling, significantly reducing total execution time. For the GPU-image, we design a simple and generic processing element (SGPE) which consists of a multiplier and an accumulator, as illustrated in Fig. 3.11. These SGPEs can be combined together to support different types of NN computation. During inference, the buffer controller regulates the NN‚Äôs structure, while the weight buffers store the model parameters. The previous intermediate results are fetched and processed by the SGPEs, and the newly computed results are sent back to the line buffers. This data exchange is efficiently managed by pipeline multiplexers. The input and output results are stored in DDR3 memory and periodically exchanged with the intermediate results buffers. This design enables flexible model inference by loading different model structures and weights in the buffer. In this prototype, we provide Yolov3, Yolov2, and Yolov2-tiny [84].

3.5.2 A Case Study

We verify the adaptability of resource control and video analytics configuration selection under the task of intelligent traffic monitoring. This case study is based on

a real-world urban traffic surveillance deployment and naturally reflects two typical dynamic factors commonly encountered in practice: (1) bandwidth dynamics, where the smart camera adjusts its frame rate based on varying network conditions; and (2) content dynamics, where objects such as vehicles, bicycles, and pedestrians appear unpredictably with diverse speeds, sizes, and densities. These real-world fluctuations demand real-time adaptation in model selection, resolution, and resource allocation, but also provide a realistic setting to demonstrate the system‚Äôs robustness and adaptability under highly dynamic conditions.

In the dynamic frame rate scenario, the smart camera adjusts its frame rate according to the available bandwidth. The edge server adapts the CPU-GPU resource allocation and selects the most suitable resolution and DNN/CNN model in response to the changing frame rates, as shown in Fig. 3.12(a). The system tends to allocate increased resources to the GPU when the frame rate is low because workloads remain manageable for the GPU compared to higher frame rate conditions.

In the dynamic content scenario, video contents change over time at a fixed frame rate of 60. There are high-speed cars in epochs 0-19, slow bikes in epochs 20-29, followed by slow cars in epochs 30-34, and fast cars in epochs 35-50. In response to these changes, the system adjusts its configuration, as shown in Fig. 3.12(b). The initial video analytics configuration is Yolov3 at 180 p for fast cars. It then switches to Yolov3 at 480 p to better handle slow bikes. This switch is because the small contents require a larger resolution for model inference. A simpler DNN/CNN model, Yolov2-tiny, is selected for the slow cars because it achieves similar accuracy for the large content compared to a more complex DNN/CNN model. Moreover, since object tracking module in the CPU-image performs better with slower-moving targets, the allocation of GPU resources is reduced

Table 3.6: Energy consumption under different FPGA configurations

Configuration	Power (W)	Throughput (FPS)	Energy/Frame (J)
Gemini+	4.65	30	0.155
CPU-image	3.61	53	0.0681
GPU-image	6.54	14	0.467

significantly.

Moreover, we measure the energy consumption of the dual-image FPGA with Monsoon Power Monitor in this case study. Specifically, we first measure the energy consumption of the overall system (Gemini+). Then we evaluate the energy consumption of the CPU-image and GPU-image respectively, where we only run a single image for video analytics. Table. 3.6 shows the power, throughput, and energy per frame for these three FPGA configurations. The CPU-image achieves the highest energy efficiency, consuming only 0.0681 J per frame. However, it can not run for a long time due to a quick accuracy drop. GPU-image incurs the highest energy cost, requiring 0.467 J per frame, due to its deep learning workload. Gemini+ balances energy and performance, achieving 30 FPS at 0.155 J per frame, 66.8% lower than GPU-image, demonstrating its ability to optimize resource allocation dynamically.

Additionally, we have measured the image switching power as 1.65 W. While the instantaneous power during switching is high, the total energy overhead is only 0.104 J due to the extremely short switching time. This results in an overall switching overhead of around 2% of the total energy consumption of Gemini+, confirming that the energy impact of dynamic switching remains negligible in practical deployment. Since FPGAs are well-known for their energy efficiency, we focus on their own energy consumption instead of comparison against other hardware such as CPU and GPU [8, 78, 107].

3.6 Chapter Summary

In this chapter, we propose Gemini+, a flexible CPU-GPU video analytics pipeline built on dual-image FPGAs to address the limitations of fixed computing resources.

The key contributions are summarized as follows:

- We propose a framework that dynamically allocates CPU and GPU resources to optimize real-time video analytics in both single-camera and multi-camera scenarios.
- We formulate two optimization problems,Äîone for each scenario,Äîand develop efficient algorithms to solve them.
- Evaluation results show that Gemini+ improves detection accuracy over fixed or GPU-dominant baselines and achieves better resource utilization than our earlier work, Gemini.
- A prototype implementation further validates the feasibility and performance benefits of the proposed design.

These findings demonstrate that flexible CPU-GPU resource control on reconfigurable hardware can significantly enhance the adaptability and effectiveness of edge video analytics systems.

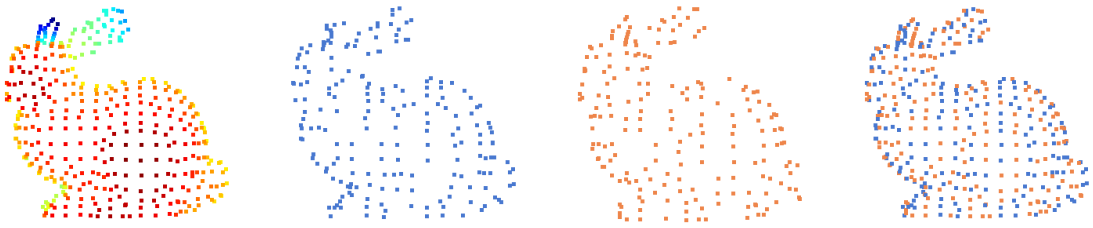
CHAPTER 4

LAYER-BASED PREFETCHING FOR VOLUMETRIC VIDEO STREAMING UNDER NETWORK DYNAMICS

4.1 Introduction

The impact of network dynamics is exacerbated by the high bandwidth consumption of volumetric videos, leading to more frequent occurrences of rebuffering and frame skipping. As a result, users are forced to endure repeated waiting periods, significantly degrading the overall immersive experience. Additionally, the fluctuating network conditions would trigger frequent quality switches during video playback, leading to an unpleasant viewing experience.

Prefetching schemes are generally believed to provide a smoother video playback experience by fetching video clips in advance with sufficient bandwidth [89, 127, 128]. However, straightforward prefetching strategies may decrease video quality due to prefetched low-quality frames. Retransmission can eliminate this by fetching high-quality versions later, but it leads to bandwidth wastage because the previously transmitted



(a) Original frame. (b) Layer 1 with 50%. (c) Layer 2 with 50% points. (d) Combined layers.

Figure 4.1: Layer partition for volumetric video.

frames are not used.

In this chapter, we propose PLVS, a novel Progressive Layer-based Volumetric Video Streaming strategy to address this issue. The core idea behind PLVS is to reuse the transmitted frames during quality improvement. One promising way is to support progressive volumetric video compression directly. However, existing codecs like Google Draco [4] and GPCC [88] do not implement this feature. To overcome this limitation, we leverage the unique data structure of point cloud-based volumetric video. This format allows us to partition the raw data into multiple disjoint subsets of points, referred to as layers. Each layer corresponds to a specific point density, representing different levels of video quality. By combining these layers, it refines video quality incrementally, as illustrated in Fig. 4.1.

We consider two practical scenarios where frames can be skipped or not. In the frame-skip scenario, frames that cannot be transmitted in time are dropped to ensure playback smoothness. In contrast, the non-frame-skip scenario requires that all frames be displayed, even if this leads to playback stalls. Although these two scenarios have different objectives, they present common challenges when designing a layer-based prefetching scheme.

First, while layer partition reduces bandwidth wastage, it brings extra transmission

(a) 12.5%-Layer with qp=8. (b) 12.5%-Layer with qp=6. (c) Combination of two layers.

Figure 4.2: Quality of single layer and combined layers. The quality of the combination of a 12.5%-layer with qp=8 and a 12.5%-layer with qp=6 is lower than a 12.5%-layer with qp=8 alone.

overhead due to lower compression efficiency, making it challenging to balance its benefits and drawbacks. To minimize bandwidth resources, the most straightforward approach is to transmit the layer with the lowest density, as it requires the least data and mitigates the impact of unstable bandwidth. However, this method results in extremely high transmission cost, because layer partitioning decreases compression efficiency.

To address this limitation, in the frame-skip scenario, we adopt a uniform layer partitioning strategy, where layers are evenly divided to maintain consistent compression efficiency. Since some frames may be skipped during transmission, a simple and lightweight partitioning approach is preferred to ensure smooth playback. In the non-frame-skip scenario, we introduce a binary tree partitioning structure, which offers layers of different densities to reduce overhead. Since every frame must be displayed, this method provides more layer partition flexibility to better adapt to dynamic network conditions.

Second, volumetric video encoding brings challenges in layer combination. Contrary to intuition, integrating compressed layers sometimes fails to improve quality and may even degrade it, as illustrated in Fig. 4.2. While combining original layers improves

quality by adding more points, encoding introduces compression loss, which can negate these gains. When the compression loss dominates the benefit of additional points, the quality will be degraded instead of improved.

To address this, we thoroughly investigate the effects of layer combination under different encoding parameters under two different scenarios. Based on these observations, we formulate a video quality optimization problem for the frame-skip scenario and a QoE optimization problem for the non-frame-skip scenario that considers the impact of encoding on layer combination.

Third, the volumetric video streaming problem becomes extremely complicated due to the introduction of prefetching techniques, layer partition, and compression quality selection. It is challenging to design an efficient algorithm to solve the problem because of its inherent nature as a combination problem.

To address this, we carefully analyze the problem and discover that the objective function in the frame-skip scenario exhibits submodular properties. This insight is crucial for designing an effective layer-based prefetching scheme. Moreover, in the non-frame-skip scenario, we propose buffer-based adaptive streaming algorithms to solve the QoE optimization problem, reducing the dependence on accurate bandwidth prediction.

In summary, our contributions are as follows:

- We propose PLVS, a progressive layer-based volumetric video streaming framework, designed to handle unpredictable network conditions for both frame-skip and non-frame-skip scenarios.
- In the frame-skip scenario, we formulate the live volumetric video streaming (LVVS) problem and the on-demand volumetric video streaming (OVVS) problem. Our investigation confirms that the objective function of video quality exhibits

submodular properties, which guides us to propose approximated algorithms.

- In the non-frame-skip scenario, we formulate a QoE optimization problem and design buffer-based algorithms to adaptively make streaming decisions based on real-time network and buffer conditions.

- We evaluate the performance of our schemes through extensive simulations under two scenarios. We compare our work with existing volumetric video streaming schemes and prefetching strategies. Results indicate that layer-based prefetching improves video quality and reduces the amount of skipped frames. Additionally, we implement a prototype to validate its feasibility in practical scenarios.

4.2 Motivation

This work is motivated by two key observations in existing prefetching schemes and volumetric video streaming solutions. First, naive prefetching strategies often fail to fully utilize bandwidth resources under dynamic network conditions. These schemes fetch future frames when bandwidth resources are sufficient to ensure smooth playback. However, bandwidth conditions can fluctuate unpredictably, and previously prefetched low-quality frames may later require upgrades if bandwidth improves. Existing prefetching schemes either do not consider quality improvement or simply replace the old frame with high-quality versions, leading to bandwidth wastage. Moreover, most existing works [27, 33, 102] focus on either point cloud density or encoding versions to support different video qualities. However, we observed that both point cloud density and encoding versions matter.

To address these issues, PLVS introduces a progressive layer-based streaming strategy. We illustrate the benefits of this approach with a toy example. We compare four different streaming strategies: (1) the non-progressive streaming strategy without re-

Table 4.1: Frame setting in the toy example

	HC-Version		LC-Version	
	Size (Mb)	Quality	Size (Mb)	Quality
Whole (100%)	1	0.6	1.5	1
Layer 1 (50%)	0.6	0.3	0.8	0.5
Layer 2 (50%)	0.6	0.3	0.8	0.5

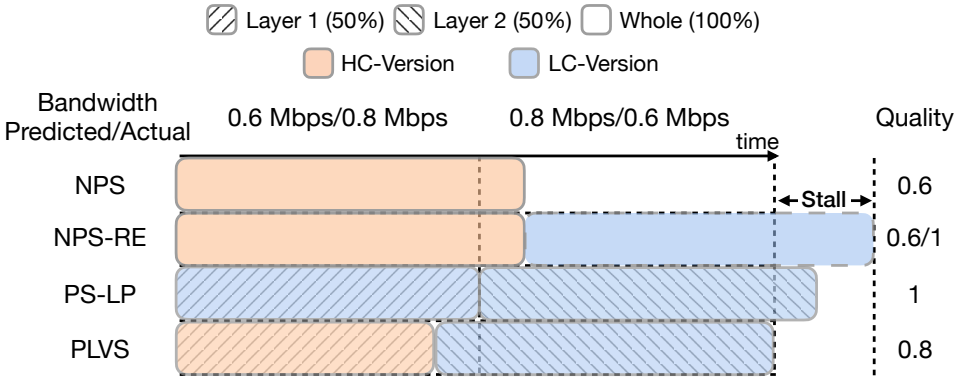


Figure 4.3: Toy example for motivation.

finement (**NPS**), (1) the non-progressive streaming strategy with refinement (**NPS-RE**), (3) the progressive streaming strategy with layer partitioning (**PS-LP**), and (4) **PLVS**, the progressive streaming strategy with multiple layers and encoding versions. For simplicity, we assume there are two time slots (e.g., 1 s) available to download a single frame, and the bandwidth remains stable in each time slot. The predicted bandwidth is 0.6 Mbps for the first time slot, while the actual bandwidth is 0.8 Mbps. In the second time slot, the predicted bandwidth is 0.8 Mbps, while the actual bandwidth is 0.6 Mbps. The frame details are listed in Table 4.1: The whole frame, containing 100% points, with high-compression encoding version (HC-Version) and low-compression encoding version (LC-Version); Layer 1 and 2, each containing 50% of the points, with HC-Version and LC-Version.

As shown in Fig. 4.3, the performance of each strategy is summarized as follows: **NPS** streams the entire frame with HC-Version during the first time slot and moves

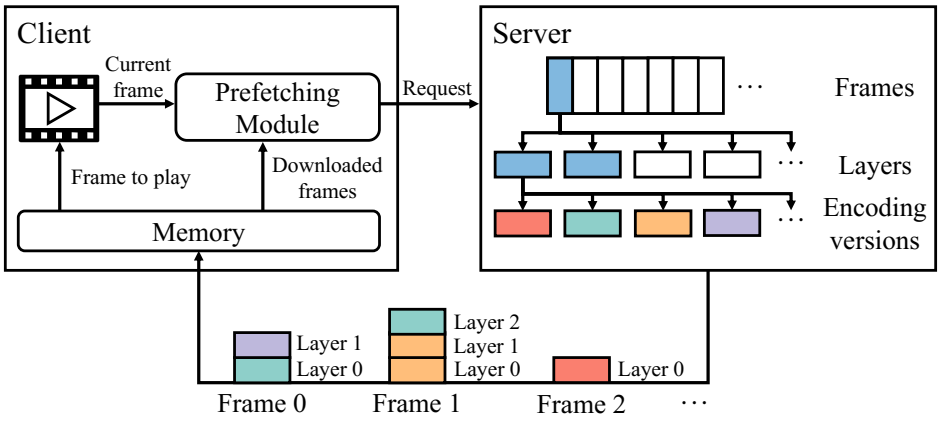


Figure 4.4: System model.

to the next frame without quality enhancement, resulting in a final frame quality of 0.6. **NPS-RE** downloads the whole frame with HC-Version during the first time slot. It either continues to transmit the next frame, leading to the same quality 0.6 as **NPS**, or refines the frame with LC-Version, resulting in a higher quality of 0.8 but a long stall. **PS-LP** transmits Layer 1 (50% of points) with LC-Version in the first slot. The positively predicted bandwidth in the second slot allows the strategy to improve the quality by transmitting Layer 2 (the remaining 50% of points) with LC-Version, leading to a short stall. **PLVS** provides another streaming decision by first transmitting Layer 1 with HC-Version, then streaming Layer 2 with LC-Version, achieving a quality of 0.8 without stall. **PLVS** can either perform like **PS-LP** if video quality is more important than playback smoothness. Overall, **PLVS** provides more streaming flexibility by enabling layer partitioning and multiple encoding versions selection, hence improving user experience in volumetric video streaming.

Table 4.2: Table of notations for PLVS for the frame-skip scenario

Variable	Description
T	Duration of the volumetric video (in seconds)
$\mathcal{T}$	Set of time slots $\{t_1, ..., t_{N_0+N}\}$
N	Number of frames in the video
$\mathcal{F}$	Set of all video frames $\{f_1, ..., f_N\}$
f_i	The i-th frame
M_i	Number of points in frame f_i
$\mathcal{P}_i$	Point cloud of frame f_i
τ_p	Playback duration of each frame
$\mathcal{L}_i$	Set of layers in frame f_i
l_{ij}	The j-th layer of frame f_i
$\mathcal{V}$	Set of encoding versions $\{v_1, ..., v_V\}$
$\mathcal{P}_{ijk}$	Reconstructed point cloud from l_{ij} with encoding v_k
$s(f_i, l_{ij}, v_k)$	Size of layer l_{ij} of f_i at version v_k
$\tau_d(f_i, l_{ij}, v_k)$	Decoding time of same
$x_t(f_i, l_{ij}, v_k)$	Portion of layer data downloaded at time t
$D(f_i)$	Playback deadline slot of frame f_i
$\mathcal{L}'_i$	Set of fully received layers for f_i
$\mathcal{P}_{\mathcal{L}'_i}$	Combined point cloud from $\mathcal{L}'_i$
$Q_f(\cdot)$	Frame quality metric
$Q(\cdot)$	Quality metric for the set of layers

4.3 PLVS Design for the Frame-Skip Scenario

4.3.1 System Model

We consider the problem of adaptive volumetric video streaming from the content server to the client, as shown in Fig. 4.4. The volumetric video is structured in layers and pre-compressed into multiple encoding versions (i.e., different encoding parameters) on the server. The client is responsible for determining the optimal contents to be (pre)fetched and sending the request to the server. The notations used in this system

model are listed in Table 4.2.

Video model. The volumetric video has the length of T seconds and contains N frames $\mathcal{F} = \{f_1, f_2, ..., f_N\}$. The frame f_i is represented as a set of M_i points $\mathcal{P}_i = \{p_{i1}, p_{i2}, ..., p_{iM_i}\}$. The playback duration of each frame is $\tau_p = \frac{T}{N}$. At the server, there are $L+1$ layers $\mathcal{L}_i = \{l_{i0}, l_{i1}, ..., l_{iL}\}$ for a frame f_i, where l_{i0} is the base layer and others are enhancement layers. Each layer l_{ij} is a subset of points in the frame, as illustrated in Fig. 4.1. For each layer, the server has V encoding versions with different encoding parameters, $\mathcal{V} = \{v_1, v_2, ..., v_V\}$. The reconstructed point cloud of the layer l_{ij} with the encoding version v_k is $\mathcal{P}_{ijk}$. The data size and decoding time of a frame f_i with layer l_{ij} and encoding version v_k are respectively denoted by $s(f_i, l_{ij}, v_k)$ and $\tau_d(f_i, l_{ij}, v_k)$.

Video streaming model. The streaming process is equally divided into slots of fixed length τ_p. There are N_0 slots for initial start-up delay, and N slots during streaming, denoted by $\mathcal{T} = \{t_1, t_2, ..., t_{N_0+N}\}$. Each slot has the bandwidth of c_t. Each frame f_i should be played before its deadline, the slot $D(f_i) = N_0 + i - 1$. At the beginning of each slot t, the client decides the data to be transmitted, which include a mix of full and partial frames, denoted as $\mathbf{x}_t = \{x_t(f_i, l_{ij}, v_k) | \forall D(f_i) < t, l_{ij} \in \mathcal{L}_i, v_k \in \mathcal{V}\}$. For example, $\mathbf{x}_t = \{x_t(f_1, l_{11}, v_1) = 1, x_t(f_2, l_{20}, v_2) = 0.3\}$ indicates that the client decides to download the full layer l_1 of frame f_1 with encoding version v_1, and 30% layer l_0 of frame f_2 with encoding version v_2. At the frame deadline $D(f_i)$, the client reconstructs and combines layers $\mathcal{L}_i^{'}$ that have been completely downloaded for display,

$$\mathcal{L}_i^{'} = \{(l_{ij}, v_k) | \sum_{t=1}^{D(f_i)-1} x_t(f_i, l_{ij}, v_k) = 1\}. \tag{4.1}$$

The received PtCl $\mathcal{P}_{\mathcal{L}_i^{'}}$ of frame f_i is the aggregation of all points in $\mathcal{L}_i^{'}$,

$$\mathcal{P}_{\mathcal{L}_i^{'}} = \bigcup_{(l_{ij}, v_k) \in \mathcal{L}_i^{'}} \mathcal{P}_{ijk}. \tag{4.2}$$

Video quality model. We measure the video quality by comparing the received PtCl

frame $\mathcal{P}_{\mathcal{L}_i^{'}}$ with the original PtCl frame $\mathcal{P}_i$. Most volumetric videos [20, 37] have color information in integers rather than floating numbers, which will not cause compression loss in Google Draco. Hence, we only measure geometry compression distortion in this work by the Point-to-point (P2Point) metric [100]. For each point p_j in the original PtCl $\mathcal{P}_i$, we identify the nearest point $p_j^{'}$ in the received PtCl $\mathcal{P}_i^{'}$ and calculate their Euclidean distance $d_{point}(p_j, p_j^{'})$. Root mean square (RMS) distance is used to calculate the P2Point distance $d_{ptcl}(\mathcal{P}_i, \mathcal{P}_i^{'})$ between the original PtCl and the received PtCl,

$$d_{ptcl}(\mathcal{P}_i, \mathcal{P}_i^{'}) = \frac{1}{M_i} \sum_{j=1}^{M_i} d_{point}(p_j, p_j^{'}). \tag{4.3}$$

Then, we calculate the distance from received PtCl to the original PtCl $d_{ptcl}(\mathcal{P}_i^{'}, \mathcal{P}_i)$ in reverse, and regard the maximum value as the final P2Point distance $d_{p2p}(\mathcal{P}_i^{'})$,

$$d_{p2p}(\mathcal{P}_i^{'}) = \max\{d_{ptcl}(\mathcal{P}_i^{'}, \mathcal{P}_i), d_{ptcl}(\mathcal{P}_i, \mathcal{P}_i^{'})\}. \tag{4.4}$$

The negative distance is used to indicate the frame quality,

$$Q_f(\mathcal{P}_i^{'}) = -d_{p2p}(\mathcal{P}_i^{'}). \tag{4.5}$$

However, we have observed that sometimes selecting a portion of the received layers can achieve better video quality than using all layers. We define the quality of the received layers as the maximum frame quality of all possible layer combinations,

$$Q(\mathcal{L}_i^{'}) = \max_{\mathcal{L}_r \in 2^{\mathcal{L}_i^{'}}} Q_f(\mathcal{P}_{\mathcal{L}_r}). \tag{4.6}$$

4.3.2 Problem Formulation

Based on the above system models, we formulate the LVVS problem and the OVVS problem in this subsection.

4.3.2.1 Live Volumetric Video Streaming Problem

In the scenario of live video streaming, the limited availability of future frames requires that each frame can be fetched just before its scheduled display time. Specifically,

we assume that each frame should be transmitted precisely Δ slots before its deadline. In this case, the LVVS problem focuses on the optimal video layers and encoding versions alone without a prefetching scheme. It aims to maximize the video quality under time constraints and is formulated as,

$$\max_{\mathbf{x}} \sum_{i=1}^{N} Q(\mathcal{L}_i^{'}) \tag{4.7}$$

$$\text{s.t. } Eq.(4.1), Eq.(4.2), Eq.(4.6) \tag{4.8a}$$

$$\sum_{x_t \in \mathbf{x}_t} x_t(f_i, l_{ij}, v_k) s(f_i, l_{ij}, v_k) \leq \tau_p c_t, \forall t \in \mathcal{T} \tag{4.8b}$$

$$\begin{aligned} &\tau_d(f_i, l_{ij}, v_k) + (t+1) \leq D(f_i), \\ &\forall t \in \mathcal{T}, x_t(f_i, l_{ij}, v_k) > 0 \end{aligned} \tag{4.8c}$$

$$\sum_{t \in \mathcal{T}} x_t(f_i, l_{ij}, v_k) \leq 1, \forall f_i \in \mathcal{F}, l_{ij} \in \mathcal{L}_i, v_k \in \mathcal{V} \tag{4.8d}$$

$$\sum_{l_{ij} \in \mathcal{L}_i, v_k \in \mathcal{V}} x_t(f_i, l_{ij}, v_k) = 0, \forall D(f_i) - t \neq \Delta \tag{4.8e}$$

$$x_t(f_i, l_{ij}, v_k) \in \{0, 1\}, \forall f_i \in \mathcal{F}, l_{ij} \in \mathcal{L}_i, v_k \in \mathcal{V}. \tag{4.8f}$$

The Constraint (4.8a) is the display rule that the subset of received layers with the highest quality is presented. The constraint (4.8b) indicates that the total size of transmitted data should not exceed the available bandwidth at each time slot. The constraint (4.8c) prevents the transmission of data that cannot be displayed in time. The constraint (4.8d) imposes that each layer with a certain encoding version can only be sent at most once. Constraints (4.8e) and (4.8f) indicate that we can only send the frame exactly Δ slots before its deadline, and hence it can not be split and transmitted across several slots.

In the following, we analyze the complexity of the LVVS problem, which is crucial for

understanding the problem challenges and guiding the algorithm design. To support the complexity analysis, we first prove that the objective function Eq. (4.7) is submodular. This submodularity allows us to reduce the LVVS problem to the budgeted submodular function maximization problem (BSFMP), which is a well-known NP-hard problem.

This problem involves selecting a subset of items from a finite set to maximize a submodular objective, subject to a cost constraint. Submodular functions capture the notion of diminishing returns, making greedy algorithms both effective and widely used in this setting. This formulation aligns well with our problem, where each candidate item corresponds to a layer-version pair, the objective is frame quality, and the cost corresponds to the data size under bandwidth constraints.

Definition 4.1. A set function $f : 2^{\mathcal{U}} \to \mathbb{R}$ is called a **submodular function** over a finite ground set $\mathcal{U}$, if for every $\mathcal{A}, \mathcal{B} \subseteq \mathcal{U}$ with $\mathcal{A} \subseteq \mathcal{B}$, the following condition holds:

$$f(\mathcal{A} \cup \{u\}) - f(\mathcal{A}) \geq f(\mathcal{B} \cup \{u\}) - f(\mathcal{B}), \tag{4.9}$$

where $u \in \mathcal{U}, u \notin \mathcal{B}$.

We prove the submodular function based on two observations. The first observation dictates the quality improvement brought by the new layer to different quality frames, while the second observation shows the consistency in layer selection across different layer sets.

Observation 4.1. *When a layer is selected for display, it enhances the quality of frames that are of lower quality more than those of higher quality.*

This observation implies that the marginal gain in quality improvement brought by a layer. This supports the diminishing returns property essential to submodular functions.

Observation 4.2. *If a layer is selected for display within a set of layers, then it should also be selected for display in any subset of that set if it belongs to the subset.*

This observation suggests consistency in layer selection. It shows that if a layer can contribute to frame quality under a large set of layers, it is expected to benefit a smaller subset.

We validate the above observations via experiments in Section 4.3.4. Leveraging these observations, we prove that the objective function (4.7) is submodular.

Theorem 4.1. *The quality function $Q(\mathcal{L}_i^{'})$ is submodular.*

Proof. The ground set of the quality function for frame f_i is $U_i = \mathcal{L}_i \times \mathcal{V} = \{(l_{ij}, v) | l_{ij} \in \mathcal{L}_i, v \in \mathcal{V}\}$. We assume that there are two sets of layers denoted by $\mathcal{A}, \mathcal{B}$ where $\mathcal{A} \subseteq \mathcal{B}$. The set of displayed layers are $\mathcal{A}^* = \arg\max_{\mathcal{L} \in 2^{\mathcal{A}}} Q_f(\mathcal{P}_{\mathcal{L}})$ and $\mathcal{B}^* = \arg\max_{\mathcal{L} \in 2^{\mathcal{B}}} Q_f(\mathcal{P}_{\mathcal{L}})$ respectively. Given a new layer $u = (l, v) \notin \mathcal{B}$, define $\mathcal{A}^{'} = \mathcal{A} \cup \{u\}, \mathcal{B}^{'} = \mathcal{B} \cup \{u\}$ as updated PtCls. Let $\mathcal{A}^{*'} = \arg\max_{\mathcal{L} \in 2^{\mathcal{A}'}} Q_f(\mathcal{P}_{\mathcal{L}})$ and $\mathcal{B}^{*'} = \arg\max_{\mathcal{L} \in 2^{\mathcal{B}'}} Q_f(\mathcal{P}_{\mathcal{L}})$ be the new optimal layer selections. Hence, $Q(\mathcal{A}) = Q_f(\mathcal{P}_{\mathcal{A}^*}), Q(\mathcal{B}) = Q_f(\mathcal{P}_{\mathcal{B}^*}), Q(\mathcal{A}^{'}) = Q_f(\mathcal{P}_{\mathcal{A}^{*'}}), Q(\mathcal{B}^{'}) = Q_f(\mathcal{P}_{\mathcal{B}^{*'}})$. Since $\mathcal{A} \subseteq \mathcal{B}$, it follows that $Q(\mathcal{A}) \leq Q(\mathcal{B})$. We consider the following 4 conditions,

- If $u \notin \mathcal{A}^{*'}$ and $u \notin \mathcal{B}^{*'}$, we have $Q(\mathcal{A}^{'}) - Q(\mathcal{A}) = Q(\mathcal{B}^{'}) - Q(\mathcal{B}) = 0$.
- If $u \in \mathcal{A}^{*'}$ and $u \notin \mathcal{B}^{*'}$, we have $Q(\mathcal{A}^{'}) - Q(\mathcal{A}) > Q(\mathcal{B}^{'}) - Q(\mathcal{B}) = 0$.
- If $u \in \mathcal{A}^{*'}$ and $u \in \mathcal{B}^{*'}$, it indicates that the new layer u can improve both quality of $\mathcal{A}$ and $\mathcal{B}$. According to the Observation 4.1 and $Q_f(\mathcal{P}_{\mathcal{A}^*}) < Q_f(\mathcal{P}_{\mathcal{B}^*})$, we have $Q_f(\mathcal{P}_{\mathcal{A}^{*'}}) - Q_f(\mathcal{P}_{\mathcal{A}^*}) \geq Q_f(\mathcal{P}_{\mathcal{B}^{*'}}) - Q_f(\mathcal{P}_{\mathcal{B}^*})$, which is equal to $Q(\mathcal{A}^{'}) - Q(\mathcal{A}) > Q(\mathcal{B}^{'}) - Q(\mathcal{B})$.
- If $u \notin \mathcal{A}^{*'}$ and $u \in \mathcal{B}^{*'}$, it means that the new layer u can improve quality of $\mathcal{B}$ but can not improve quality of $\mathcal{A}$, which is contrary to the Observation 4.2.

Hence, we have $Q(\mathcal{A} \cup \{u\}) - Q(\mathcal{A}) \geq Q(\mathcal{B} \cup \{u\}) - Q(\mathcal{B})$, which indicate that the quality function $Q(\mathcal{L}_i^{'})$ is submodular. ■

Theorem 4.2. *The LVVS problem is NP-hard.*

Proof. We prove this theorem by reducing the LVVS problem to the BSFMP. Due to the constraint (4.8e) that only the frame exactly Δ slots before its deadline can be transmitted, the original objective function $\sum_{i=1}^{N} Q(\mathcal{L}_i^{'})$ is equal to maximize the quality function of each frame $Q(\mathcal{L}_i^{'})$ independently.

For each subproblem $\max_{\mathbf{x}_t} Q(\mathcal{L}_i^{'})$, we define the domain of elements as $u_i = \mathcal{L}_i \times \mathcal{V}$. Each element in the domain has a cost (size) $s(f_i, l_{ij}, v_k)$. The problem then transforms into optimizing the submodular function $Q(\mathcal{L}_i^{'})$ subject to a cost constraint, specifically the budget constraint (4.8b).

In this way, the LVVS problem is equivalent to a series of BSFMP, which is NP-hard [49]. ■

As the LVVS problem is NP-hard, it is computationally infeasible to solve using exhaustive search methods for large instances. Therefore, it requires an efficient algorithm to solve the LVVS problem. In the LVVS problem, a key trade-off in layer selection under constrained conditions lies in balancing playback smoothness and visual quality. Streaming lower-quality layers helps maintain real-time playback but degrades visual experience, whereas selecting higher-quality layers improves video quality at the risk of frame drops due to limited bandwidth resources.

4.3.2.2 On-demand Volumetric Video Streaming Problem

In the scenario of on-demand video streaming, the content server contains the entire video file rather than just limited future frames. Consequently, the determination extends

beyond selecting layer qualities. It also involves frames to prefetch. The goal of the OVVS problem is to devise an optimal frame transmission scheme $\mathbf{x}$ to maximize the overall video quality:

$$\max_{\mathbf{x}} \sum_{i=1}^{N} Q(\mathcal{L}_i^{'}) \tag{4.10}$$

$$\text{s.t. } (4.8a), (4.8c), (4.8b), (4.8d) \tag{4.11a}$$

$$x_{t,f,l,v} \in [0,1], \tag{4.11b}$$

The primary differences between the OVVS and LVVS problems lie in the constraints (4.8e), (4.8f), and (4.11b). These differences indicate that in the OVVS problem, layers can be transmitted at any time before their deadline and can be divided across multiple slots.

The prefetching feature of the OVVS problem introduces additional complexity compared to the LVVS problem. Since the LVVS problem has been proven to be NP-hard, the OVVS problem inherits this computational intractability, making it challenging to solve efficiently.

Theorem 4.3. *The OVVS problem is NP-hard.*

Proof. The OVVS problem can be reduced to the LVVS problem by requiring that frames are transmitted only for the upcoming slot. Hence, the OVVS problem is NP-hard. ■

Therefore, it is essential to design an algorithm that can efficiently find a solution to the OVVS problem effectively, balancing computational feasibility and solution quality. In the OVVS problem, the trade-off becomes more complex because it includes prefetching beyond layer selection. In addition to deciding which quality layer to stream, the system must also determine which frames to prefetch in advance, under limited bandwidth

and computing constraints. Prefetching too many future frames can lead to resource exhaustion and quality decreases, while conservative prefetching may result in missing frames during playback. Therefore, the trade-off involves jointly optimizing layer quality and prefetching range to balance bandwidth usage, latency, and video quality.

4.3.3 Algorithms

In this subsection, we analyze the LVVS problem and propose an approximation algorithm for it. Additionally, we study the OVVS problem and propose an online algorithm.

4.3.3.1 The Algorithm for LVVS Problem

Since the LVVS problem does not involve prefetching future frames, we can decompose it into a series of one-slot problems. Each one-slot problem aims at maximizing the frame quality in that slot. Our primary focus in each slot is to determine the optimal data request for a specific frame.

As discussed earlier, each one-slot problem can be reduced to a BSFMP. Inspired by the well-established greedy algorithm for the BSFMP, we propose a Single Frame Greedy Streaming (SFGS) Algorithm to solve the one-slot problem. The algorithm employs two greedy search rules: the quality-based greedy rule (lines 1-11) and the cost-aware greedy rule (lines 12-22). In the quality-based greedy rule, the algorithm identifies and selects the layer that contributes the most to the overall quality. Conversely, the cost-aware greedy rule prioritizes the layer with the highest quality-to-size ratio. Ultimately, the algorithm returns the solution that yields the higher quality among these two greedy approaches.

The SFGS algorithm provides an efficient solution for the LVVS problem with a time complexity of $O(|\mathcal{U}|^2)$. It is dominated by two iterative greedy selection processes. Each

Algorithm 4.1 Single Frame Greedy Streaming Algorithm

Input: Ground set $\mathcal{U} = \mathcal{L} \times \mathcal{V}$, submodular function $Q(\cdot)$, bandwidth budget c, slot length τ_p, decoding time function $\tau_d(\cdot)$, the number of ahead slots Δ, current frame f

Output: The request decision in the slot $\mathbf{x}_t = \{x_t(f, l_0, v_1), x_t(f, l_0, v_2), ..., x_t(f, l_L, v_V)\}$

$\mathcal{L}_q \leftarrow \emptyset$, $c_q \leftarrow c\tau_p$, $\mathcal{U}_q \leftarrow \mathcal{U}$
while $\mathcal{U}_q \neq \emptyset$ **do**
  foreach $u \in \mathcal{U}_q$ **do**
    if $s(f,u) > c$ *or* $\tau_d(f,u) > \Delta\tau_p$ **then**
      Delete u from $\mathcal{U}_q$
    end
  end
  $u_q^* \leftarrow \arg\max_{u \in \mathcal{U}_q} Q(\mathcal{L}_q \cup \{u\})$
  $c \leftarrow c - s(f, u_q^*)$, $\mathcal{L}_q \leftarrow \mathcal{L}_q \cup \{u_q^*\}$
  Delete u_q^* from $\mathcal{U}_q$
end
$\mathcal{L}_c \leftarrow \emptyset$, $c_c \leftarrow c$, $\mathcal{U}_c \leftarrow V$
while $\mathcal{U}_c \neq \emptyset$ **do**
  foreach $u \in \mathcal{U}_c$ **do**
    if $s(f,u) > c$ *or* $\tau_d(f,u) > \Delta\tau_p$ **then**
      Delete u from $\mathcal{U}_c$
    end
  end
  $u_c^* \leftarrow \arg\max_{u \in \mathcal{U}_c} \frac{Q(\mathcal{L}_c \cup \{u\})}{s(f,u)}$
  $c \leftarrow c - s(f, u_c^*)$, $\mathcal{L}_c \leftarrow \mathcal{L}_c \cup \{u_c^*\}$
  Delete u_c^* from $\mathcal{U}_c$
end
$\mathcal{L}' \leftarrow \arg\max_{\mathcal{L} \in \{\mathcal{L}_q, \mathcal{L}_c\}} Q(\mathcal{L})$
foreach $u \in \mathcal{L}'$ **do**
  $x_t(f,u) = 1$
end
return request decision for in the slot $\mathbf{x}_t$

greedy algorithm has a time complexity of $O(|\mathcal{U}|^2)$. In practical scenarios, the complexity can be reduced to $O(|\mathcal{U}|)$ due to early termination. For example, when bandwidth and/or latency budgets do not support more layers for transmission, a large portion of the candidate layer in the set $\mathcal{U}$ will be filtered out. Therefore, the iteration of the SFGS algorithm terminates earlier, which reduces the overall complexity. Moreover, the SFGS algorithm achieves an approximation factor of $\frac{1}{2} \cdot (1 - \frac{1}{e})$, ensuring the solution performance.

Theorem 4.4. *The algorithm for LVVS problem achieves an approximation factor of* $\frac{1}{2} \cdot (1 - \frac{1}{e})$.

Proof. Let $\mathcal{L}^*$ denote the optimal layer set solution for the problem. The Alg. 4.1 generates a sequence of layer sets by the cost-aware greedy rule in each iteration, denoted by $\mathcal{L}_{cj} = \{u_{c1}, u_{c2}, ..., u_{cj}\}$. Let u_{ck+1} be the first layer which is selected in the optimal layer set $\mathcal{L}^*$ but not added in to the cost-aware layer set $\mathcal{L}_c$, because it violates the budget, i.e, $\sum_{h'=1}^{k+1} s(u_{ch'}) > c\tau_p$. According to the lemmas from prior works [49, 50], for all $j = 1, 2, ..., k+1$,it holds that,

$$Q(\mathcal{L}_{cj}) \geq [1 - \prod_{h=1}^{i} (1 - \frac{s(u_{ch})}{c\tau_p})]Q(\mathcal{L}^*). \tag{4.12}$$

Since $\sum_{h'=1}^{k+1} s(u_{ch'}) > c\tau_p$, it follows that,

$$\begin{aligned} Q(\mathcal{L}_{ck+1}) &\geq [1 - \prod_{h=1}^{i} (1 - \frac{s(u_{ch})}{\sum_{h'=1}^{k+1} s(u_{ch'})})]Q(\mathcal{L}^*) \\ &\geq [1 - (1 - \frac{1}{l+1})^{l+1}]Q(\mathcal{L}^*) \\ &\geq (1 - \frac{1}{e})Q(\mathcal{L}^*). \end{aligned} \tag{4.13}$$

Note that $Q(\mathcal{L}_{ck+1}) - Q(\mathcal{L}_{ck})$ is bounded by $Q(\{u^*\})$, where $u^* = \arg\max_{u \in \mathcal{L} \times \mathcal{V}} Q(\{u\}), s(u^*) \leq c\tau_p, \tau_d(u^*) \leq \Delta\tau_p$. That is, $u^* \in \mathcal{L}_q$ is the first layer in the quality-based layer set. Hence,

$$Q(\mathcal{L}_{ck}) + Q(\{u^*\}) \geq Q(\mathcal{L}_{ck+1}) \geq (1 - \frac{1}{e})Q(\mathcal{L}^*). \tag{4.14}$$

Either $Q_{\mathcal{L}_{ck}}$ or $Q(\{u^*\})$ is greater than or equal to $\frac{1}{2}(1 - \frac{1}{e})Q(\mathcal{L}^*)$. Since $Q(\mathcal{L}_c) \geq Q(\mathcal{L}_{ck})$ and $Q(\mathcal{L}_q) \geq Q(\{u\})$, the algorithm achieves an approximation factor of $\frac{1}{2}(1 - \frac{1}{e})$. ■

4.3.3.2 The Algorithm for On-demand Volumetric Video Streaming Problem

We first propose an offline algorithm for the OVVS problem, assuming that future bandwidth information is available. Then, we introduce an online algorithm to handle scenarios where bandwidth information is unavailable.

We regard the OVVS algorithm as a two-step problem: The first step involves allocating bandwidth for each frame, while the second step focuses on selecting the optimal layers and encoding versions for each frame given the allocated bandwidth. The second-stage problem can be addressed using Alg. 4.1. Therefore, our primary concern is the bandwidth resource allocation in the first stage.

We simplify the problem based on two key observations. The first observation notes the similarities among frames sharing the same layer and encoding version. The second observation assumes that the frame quality with a bandwidth budget can be regarded as a concave function. Utilizing these insights, we develop both online and offline algorithms to efficiently address the problem.

Observation 4.3. *Data size and decoding time for frames with identical layers and encoding versions are similar, while decoding duration for any layer across different versions does not exceed the length of a slot.*

Observation 4.4. *The frame quality with bandwidth budget can be approximately regarded as a concave function.*

Based on the above observations (detailed in Section 4.3.4), we assume that all frames are identical, i.e., $s(f_i, l_j, v_k) = s(f_{i'}, l_j, v_k)$ and $\tau^d(f_i, l_j, v_k) = \tau^d(f_{i'}, l_j, v_k)$. Considering the decoding time, we require that the frame be downloaded one slot before its deadline. The constraint (4.8c) is relaxed as $t \le D(f_i)$. Hence, the bandwidth allocation problem is formulated as follows,

$$\max_{C_f} \sum_{i=1}^{N} Q^*(c_{f_i}) \tag{4.15}$$

$$\text{s.t.} \sum_{i=1}^{N} c_{f_i} \le \sum_{j=1}^{t_i^{ddl}-1} c_i, \forall 1 \le i \le N. \tag{4.16}$$

The bandwidth allocation algorithm, as outlined in Alg. 4.2, strategically distributes bandwidth across each frame to optimize video quality. The algorithm balances the

Algorithm 4.2 Backward Bandwidth Allocation Algorithm

Input: Bandwidth list $C = \{c_1, c_2, ..., c_{N+N_0-1}\}$

Output: Allocated bandwidths for each frame $C^* = \{c_1^*, c_2^*, ..., c_N^*\}$

1 $c_0 \leftarrow \sum_{i=1}^{N_0-1} c_i$
2 $C^* \leftarrow \{c_0\} \cup (C - \{c_1, c_2, ..., c_{N_0-1}\})$
3 **for** $i \leftarrow N + N_0$ ***to*** 1 **do**
4 $c_{total} \leftarrow \sum_{j=i}^{N+N_0} c_i$, $c \leftarrow \frac{c_{total}}{N+N_0-i+1}$
5 **if** $c > c_i^*$ **then**
6 **for** $j \leftarrow N + N_0$ ***to*** i **do**
7 **if** $c > c_j^*$ **then**
8 $c_j^* \leftarrow c$
9 $c_{total} \leftarrow c_{total} - c_j^*$, $c \leftarrow \frac{c_{total}}{N+N_0-i+1}$
10 **end**
11 **end**
12 **return** Allocated bandwidths C^*

bandwidth backward, because the bandwidth after the deadline can not be used to transmit previous frames. It begins by aggregating the bandwidth from the start-up delay into a virtual slot. Then, from the last frame to the first frame, it allocates bandwidth to each frame in reverse order (lines 3-11). During each allocation step, the algorithm compares the expected average bandwidth to the currently allocated bandwidth for the frame. If the average bandwidth is less than or equal to the current allocation, it indicates insufficient bandwidth addition from the new slot. In this scenario, the algorithm skips further reallocation for that frame and proceeds to the previous frame without changes (line 5). Otherwise, it updates the bandwidth for this frame as the average bandwidth (line 8). After each reallocation, the algorithm recalculates the total bandwidth and average bandwidth (line 9). The process repeats until all frames have received their bandwidth assignments.

To theoretically evaluate the effectiveness of the backward bandwidth allocation algorithm, we analyze its optimality under specific conditions, as the above two observations indicate. This analysis highlights the critical assumptions required for the algorithm to

guarantee an optimal solution.

Theorem 4.5. *The backward bandwidth allocation achieves the optimal solution for the optimization problem* (4.15)*, under the conditions*

$$s_{f,l,v} = s_{f',l,v}, \tau^d_{f,l,v} = \tau^d_{f',l,v}, \forall f, f' \in \mathcal{F},$$

$$\frac{Q^*(c_1) + Q^*(c_2)}{2} \leq Q^*(\frac{c_1 + c_2}{2}).$$

Proof. Given that all frames are identical in terms of size and decoding time across layers and versions, each frame will achieve the same quality level if the allocated bandwidth is consistent. Meanwhile, the backward bandwidth allocation solution ensures that $c_{f_i} \leq c_{f_j}, \forall i < j$. If this were not the case, the bandwidth between f_i and f_j would be averaged. As a result, the bandwidth differences among frames can not be reduced further. Due to $\frac{Q^*(c_1)+Q^*(c_2)}{2} \leq Q^*(\frac{c_1+c_2}{2})$, it achieves the optimal solution. ■

While Theorem 4.5 confirms the optimality of the backward bandwidth allocation strategy under the given conditions, its practical applicability depends on its computational efficiency. The time complexity of this algorithm is $O((N + N_0)^2)$, where N represents the number of frames and N_0 denotes the start-up delay. As N_0 is typically a constant in practical scenarios, the complexity simplifies to $O(N^2)$. Based on this algorithm, we propose offline and online algorithms for the OVVS problem.

An offline algorithm to address the OVVS problem is proposed in Alg. 4.3. Given the future bandwidth information, the process is initiated by allocating optimal bandwidth for each frame using Alg. 4.2 (line 1). Then, it determines the appropriate layers and encoding versions for each frame within the given bandwidth constraints by Alg. 4.1 (line 4). Finally, the algorithm downloads these layers slot by slot (lines 7-15).

Algorithm 4.3 Offline Prefetch-enabled Streaming Algorithm

Input: Bandwidth list $C = \{c_1, c_2, ..., c_{N+N_0-1}\}$, Ground set $\mathcal{U} = \mathcal{L} \times \mathcal{V}$, submodular function Q, slot length τ_p

Output: The request decision in each slot $\mathbf{x} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_{N+N_0-1}\}$

$C_f \leftarrow$ The bandwidth for each frame by Alg. 4.2
$t \leftarrow 1$
for $i \leftarrow 1$ ***to*** N **do**
  $\mathbf{x}_{f_i} \leftarrow$ The layers and encoding versions for the frame with the bandwidth budget c_{f_i} by Alg. 4.1
  $c^{'} \leftarrow$ The size of all requested data based on $\mathbf{x}_{f_i}$
  $c_t^{'} \leftarrow$ Bandwidth in current slot c_t
  foreach $u \in \{u | x_{f_i}(u) = 1\}$ **do**
    $s_u \leftarrow s(f_i, u), s_u^{'} \leftarrow \min\{s_u, c_t^{'}\}, x_t(f_i, u) \leftarrow \frac{s_u^{'}}{s(f,u)}$
    $c_t^{'} \leftarrow c_t - s_u^{'}, s_u \leftarrow s_u - s_u^{'}$
    while $s_u > 0$ **do**
      $t \leftarrow t + 1$
      $c_t^{'} \leftarrow c_t, s_u^{'} \leftarrow \min\{s_u, c_t^{'}\}, x_t(f_i, u) \leftarrow \frac{s_u^{'}}{s(f,u)}$
      $c_t^{'} \leftarrow c_t - s_u, s_u \leftarrow s_u - s_u^{'}$
    end
  end
end
return The request decision $\mathbf{x}$

The offline algorithm has a time complexity of $O(N \times |\mathcal{U}|^3)$ or $O(N^2)$, determined by the two main components: the bandwidth allocation component and the layer transmission component. The bandwidth allocation, computed by the Alg. 4.2, achieves the time complexity of $O(N^2)$, while the layer transmission component incurs a time complexity of $O(N \times |\mathcal{U}|^3)$. When the number of frames N significantly exceeds the number of possible layers $|\mathcal{U}|$, for example, in the case of an extremely long video, the time complexity of this offline algorithm becomes $O(N^2)$. Conversely, when the values of these two variables are similar, the overall complexity of the algorithm is dominated by the layer transmission component, resulting in a complexity of $O(N \times |\mathcal{U}|^3)$.

However, in real-world scenarios, future bandwidth information is often unavailable. To address this challenge, we adopt Model Predictive Control (MPC) [83] in our online

Algorithm 4.4 Online Prefetch-enabled Streaming Algorithm

Input: Ground set $\mathcal{U} = \mathcal{L} \times \mathcal{V}$, submodular function Q, decoding time budget τ, parameter k

Output: The request decision in each slot $\mathbf{x} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_{N+N_0-1}\}$

1 **for** $t \in \mathcal{T}$ **do**
2 $\quad C_t^k \leftarrow$ predict bandwidth in slot $[t, t+k]$
3 $\quad \mathbf{x}_t^{t+k} \leftarrow$ solve the maximization problem $\sum_{i=t-N_0}^{t-N_0+k} Q(P_i)$ with C_t^k and $\mathbf{x}$ by Alg. 4.3
4 $\quad x_t \leftarrow x_t$ from $\mathbf{x}_t^{t+k}$
5 **end**
6 **return** The request decision in each slot $\mathbf{x}$

streaming algorithm, as shown in Alg. 4.4. The algorithm predicts the bandwidth in slots $[t, t+k]$ based on past bandwidth data (line 2), and solves an optimization problem at each time slot to maximize the video segment quality in the future slots $[t, t+k]$ (line 3), denoted as $\sum_{i=t-N_0}^{t-N_0+k} Q(P_i)$. The decisions for slots other than t are dropped (line 4). The process iterates until decisions for all slots are determined.

For each subproblem with k slots, the computational complexity of this online algorithm is $O(k \times |\mathcal{U}|^3)$ or $O(k^2)$. The actual complexity depends on the relative size of k and $|\mathcal{U}|$, as analyzed in the offline algorithm. Since the algorithm determines the transmission in one slot at each time until the end of the video, the overall complexity scales as $O(N \times k \times |\mathcal{U}|^3)$ or $O(N \times k^2)$. In most cases, k can be regarded as a small constant, which simplifies the overall complexity to $O(N \times |\mathcal{U}|^3)$ or $O(N)$.

The online algorithm can significantly reduce computational complexity compared to the offline algorithms, particularly for long videos. In such cases, the number of frames N is far beyond the number of possible layers $|\mathcal{U}|$. Hence, the offline algorithm grows quadratically with N, while the online algorithm grows linearly. In contrast, when N and $|\mathcal{U}|$ are of similar magnitude, both algorithms share the same time complexity of $O(N \times |\mathcal{U}|^3)$. However, the online algorithm benefits from processing one slot at a time, avoiding the need to evaluate all N frames simultaneously. This slot-by-slot approach not

only reduces implementation complexity but also enhances adaptability under network dynamics, making it more practical for real scenarios.

4.3.4 Evaluation

In this section, we evaluate the performance of two algorithms through simulations to answer the following questions:

- How do different layers and encoding versions affect video streaming? (Section 4.3.4.2)
- How does the layer-based greedy streaming algorithm compare to other volumetric video streaming strategies in live volumetric video streaming? (Section 4.3.4.3)
- How does the layer-based prefetching scheme compare to other volumetric prefetching schemes in on-demand volumetric video streaming? (Section 4.3.4.4)

Furthermore, we present a case study in Section 4.3.4.5 to provide a detailed explanation of the layer-based prefetching scheme.

4.3.4.1 Experiment Setup

Dataset. We use the 8i Voxelized Full Bodies (8iVFB v2) [20], the CMU Panoptic dataset [45], and the Microsoft Voxelized Upper Bodies dataset [14] as volumetric video resources. The 8iVFB v2 dataset consists of high-resolution volumetric video sequences of individual people. Each sequence lasts for 10 seconds with 300 frames. The CMU Panoptic dataset captures people activities in a large-scale space, and hence these videos are of low point cloud density. The Microsoft Voxelized Upper Bodies Dataset focuses on the upper-body region of individual people with two resolutions $1024 \times 1024 \times 1024$ and $512 \times 512 \times 512$. Since the resolution of the 8i Voxelized Full Bodies Dataset is

Table 4.3: Video information

	8iVFB v2	**CMU Panoptic**	**Microsoft Voxelized Upper Bodies**
Video Contents	Single person	Multiple people	Upper-body region of individual people
Selected Sequence	Longdress	Toddler	Andrew
Video Duration (Second)	10	54	10
Average Point Cloud Density (K/Unit Volume)	1108	86	90
Average Number of Points (K/Frame)	834	225	284

$1024 \times 1024 \times 1024$, we use the sequences of a different resolution, $512 \times 512 \times 512$, from it. We select one representative video sequence from each dataset for evaluation, summarized in Table 4.3.

Communication Resource. In the simulations, we select three network traces from a 5G Dataset [80]. The first trace has an average bandwidth of 15 Mbps, the second trace has an average bandwidth of 30 Mbps, and the third trace has an average bandwidth of 60 Mbps.

Algorithm overhead. The single frame greedy streaming algorithm operates in no more than 4 ms, while the online prefetch-enabled streaming algorithm takes no more than 6 ms. These overheads are negligible when compared to the time required for video streaming.

Evaluation Metrics. We measure the volumetric video quality by using the average Point-to-Point distance between the encoded video and the original video as indicated in Section 4.3.1. Generally, the video quality improves with the data size. Otherwise, we transmit the one with high quality and low size.

Table 4.4: Simulation parameters

Variable	Description	Default Value (Range)
$\mathcal{L}_i$	The set of uniformly partitioned layers for each frame i	$\mathcal{L} = \{l_{i0}, l_{i1}, l_{i2}, l_{i3}\}$
l_{ij}	The layer j for the frame i (point cloud density)	25%
$\mathcal{V}$	The set of quantization parameters for encoding	$\mathcal{V} = \{5, 7, 9, 11, 13\}$
N_0	Start-up delay (slots)	5
k	Bandwidth prediction window size (slots)	4

Parameter settings. The parameters are listed in Table 4.4. Each volumetric video frame i is uniformly partitioned into 4 layers, denoted as $\mathcal{L}i = \{l_{i0}, l_{i1}, l_{i2}, l_{i3}\}$. Hence, each layer l_{ij} contains 25% point cloud density. Layers are encoded using Google Draco with quantization parameters selected from $\mathcal{V} = \{5, 7, 9, 11, 13\}$. These encoding parameters ensure diverse video quality. A start-up delay of $N_0 = 5$ slots, which is short enough to minimize user-perceived latency while still providing sufficient time for initial buffering. Additionally, the bandwidth prediction window size is set to $k = 4$ slots by default. It achieves a suitable balance between prediction accuracy and computational efficiency. These settings are designed to reflect realistic streaming scenarios, enabling a comprehensive evaluation of the proposed algorithms.

Baselines. We evaluate the proposed layer-based prefetching scheme against multiple baselines under two scenarios, the live volumetric video streaming and on-demand volumetric video streaming. In the live volumetric video streaming scenario, prefetching is not enabled since videos are generated and transmitted at the same time. Hence, we compare the SFGS algorithm with the following three baselines.

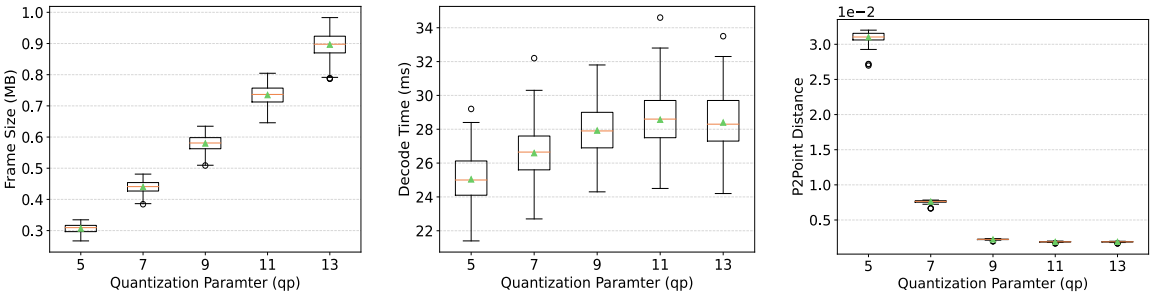
- **Non-adaptive streaming (NAS).** This scheme transmits the video content at a constant bit rate throughout the entire session, without considering the network conditions. It would skip frames when it is not able to transmit the frame in time. We set the quantization parameter in this scheme as $v = 9$.
- **Non-layer adaptive streaming (NLAS).** This scheme is similar to adaptive 2D

video streaming [93, 118], which transmits video with different qualities (encoding version) based on the available bandwidth.

- **Single-quantization layer-based streaming (SQLS).** This scheme is similar to the SVC streaming in 2D videos [19, 41], which improves video quality by adding new layers. It provides only one encoding version for videos, and it enables layer-based transmission. We set the quantization parameter in this scheme as $v = 9$.

In the on-demand volumetric video streaming scenarios, we compare our online and offline algorithms against the above streaming schemes with the following prefetching strategies.

- **BOLA** [92] is a classical buffer-based prefetching strategy that determines video quality based on current buffer states. However, once frames are transmitted, their quality cannot be upgraded, as BOLA does not support the retransmission of previously transmitted data.
- **BOLA-FS** [91] improves BOLA by allowing retransmission of prefetched frames. When bandwidth improves, high-quality versions of frames can be transmitted for quality upgrades. However, since it is not designed for layer-based videos, previously transmitted frames cannot be combined with the new versions. Therefore, low-quality versions are discarded, leading to bandwidth wastage.
- **LBP** [19] is a prefetching scheme designed for 2D SVC videos. Videos are encoded into different layers, which can be combined to achieve higher quality. This enables progressive quality upgrades by transmitting and combining layers as network bandwidth improves. However, unlike pixel-based videos, where each layer directly represents a quality level, the quality of volumetric videos depends on the number



(a) Encoded size. (b) Decode time. (c) Frame quality.

Figure 4.5: Impact of quantization parameters.

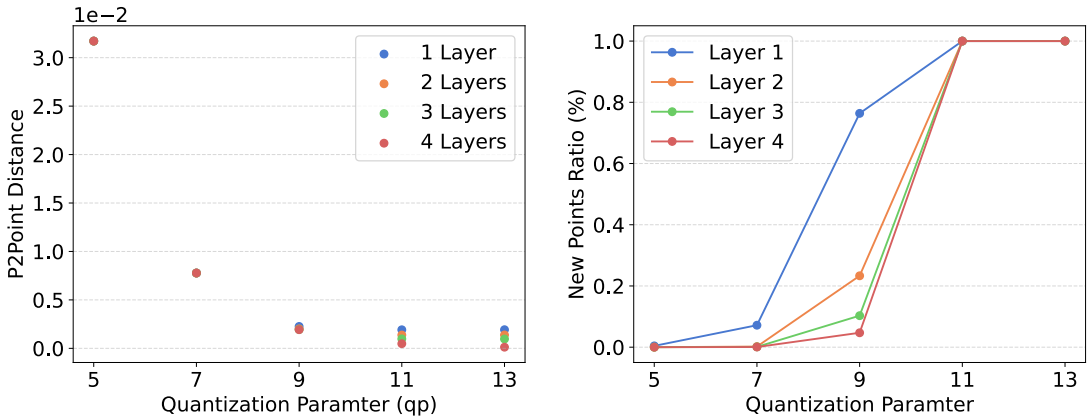
of points in each layer and the compression parameters. Hence, this prefetching scheme cannot be directly applied to volumetric video streaming.

4.3.4.2 The Impact of Layers and Encoding Versions

We conduct experiments with different layers and encoding versions to understand their influence on video streaming performance. These results help validate the observations discussed in Section 4.3.1 and 4.3.3.

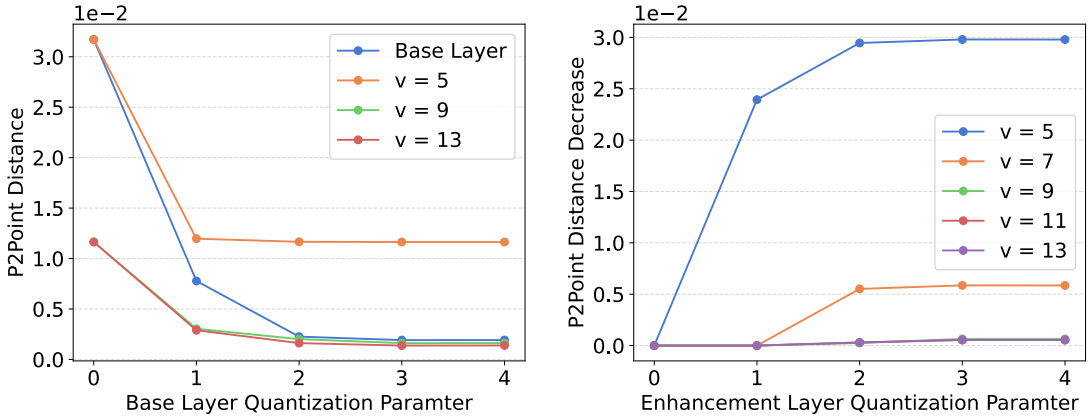
Impact of quantization parameters. Fig. 4.5 illustrates the effects of varying quantization parameters on encoded data size, decoding time, and frame quality for a single layer. With the increase of quantization parameters, the encoded data size, decoding time, and frame quality increase as well. The box plots demonstrate that frames from the same video with the same quantization parameters exhibit similar characteristics in terms of size, decoding time, and quality, justifying the assumption of identical frame properties within our algorithms. Although there is a broader range in decoding times, they are almost less than the duration of one frame playback (33 ms). Consequently, we adopt this frame duration as the decoding time budget in our algorithms.

Impact of layers. We first study the impact of layers with the same quantization parameters, as shown in Fig. 4.6. When using low quantization parameters (e.g., $v = 5$),



(a) Frame quality. (b) Number of new points.

Figure 4.6: Impact of layers with the same quantization parameter.

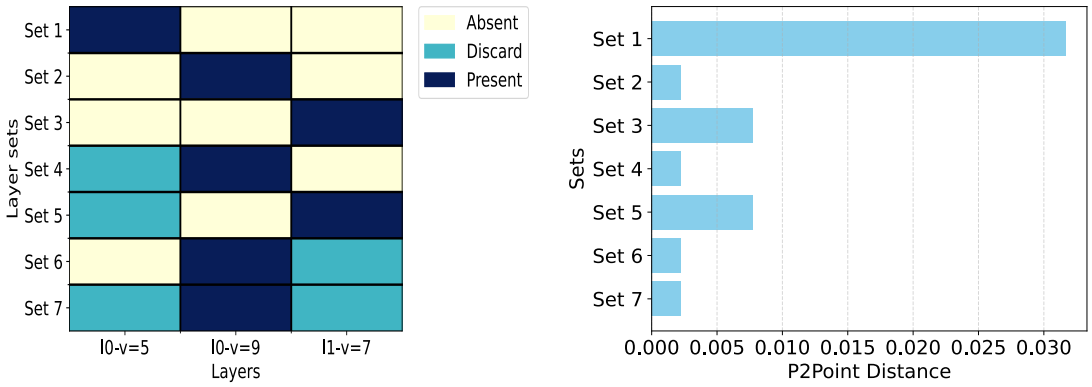


(a) Layer addition. (b) Quality improvement.

Figure 4.7: Impact of layers with different quantization parameters.

adding new layers results in only marginal improvements in video quality, as indicated in Fig. 4.6(a). This is attributed to the minimal addition of distinct points in new layers at such low encoding versions, as illustrated in Fig. 4.6(b). For example, when $v = 5$, there are virtually no new points in the second, third, and fourth layers. When $v = 11, 13$, the points in the new layers are completely different from the previous layers.

In Fig. 4.7, we analyze the video quality of layers with different quantization parameters. We consider the simplest scenario where only two layers are available. Fig. 4.7(a) illustrates that merely combining two layers may not consistently yield high video quality.



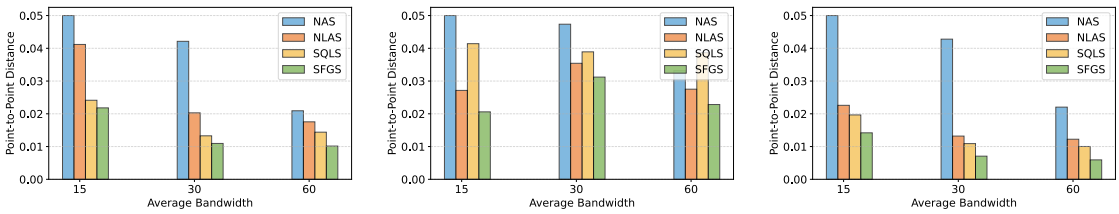
(a) Layer selection.

(b) Optimal frame quality.

Figure 4.8: Layer selection in different layer sets.

For example, combining the enhancement layer with $v = 5$ with any other base layers of higher encoding parameters leads to an increase in P2Point distance. This occurs because the enhancement layer with low encoding parameters brings more compression loss than point density benefit. On the other hand, Fig. 4.7(b) compares the P2point distance decrease (quality improvement) when we combine the same enhancement layer with different base layers. It reveals that the enhancement layer contributes more to base layers with lower encoding parameters. With the same enhancement layer, the decrease in Point-to-Point distance is greater with lower encoding versions of the base layer, aligning with Observation 4.1. Moreover, the enhancement layers with lower encoding parameters, such as $v = 5$, offer minimal benefits, as the high compression loss outweighs the additional points benefit. Conversely, layers with higher encoding parameters, like $v = 13$, lead to a dramatic improvement in video quality.

In Fig. 4.8, we validate the Observation 4.2 via a simple example where there are three layers to select: two base layers with encoding version $v = 5, 9$, one enhancement layer with encoding version $v = 7$. Each layer has three states: absent, discard, and present, as illustrated in Fig. 4.8(a). It is observed that if a layer is selected in a larger set, it is consistently chosen in all its subsets. For instance, the base layer with $v =$



(a) Longdress. (b) Toddler. (c) Andrew.

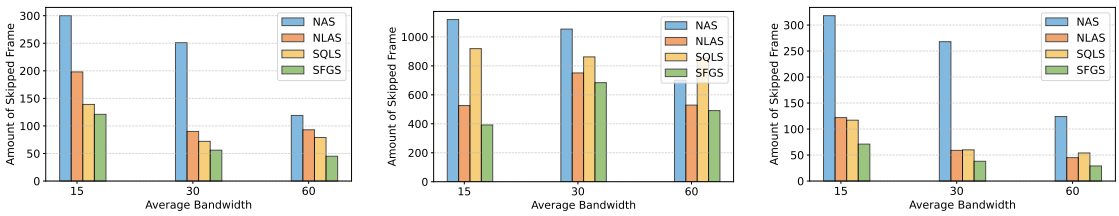
Figure 4.9: The average video quality of different streaming schemes in the live volumetric video scenario.

9 is selected in set 7 ($\{(l_0,5),(l_0,9),(l_1,7)\}$) and is also selected in all its subsets: set 6 ($\{(l_0,9),(l_1,7)\}$), set 4 ($\{(l_0,5),(l_0,9)\}$) and set 2 ($\{(l_0,9)\}$). The qualitative impact of these selections on the frame quality is detailed in Figure 4.8(b). The frame quality demonstrates that optimal layer selection is complex. The impact of the frame layer and encoding version should be jointly considered while streaming. For example, combining the base layer with encoding version $v = 9$ and the enhancement layer with encoding version $v = 7$ surprisingly results in lower video quality compared to using the single base layer at $v = 9$ alone.

4.3.4.3 PLVS Performance in Live Scenarios

We evaluate the performance of the SFGS algorithm in the live volumetric video streaming scenario to highlight the benefits of the layer-based design. Although prefetching is not applicable in live streaming, the layer-based design enhances streaming flexibility by jointly optimizing the number of points and compression parameters for each frame. This flexibility allows the algorithm to adapt to varying bandwidth conditions, improving streaming performance.

We evaluate the performance of different schemes in terms of the average video quality and the amount of skipped frames, as illustrated in Fig. 4.9 and Fig. 4.10. To facilitate comparison, we assign a penalty point-to-point distance of 0.05 for skipped

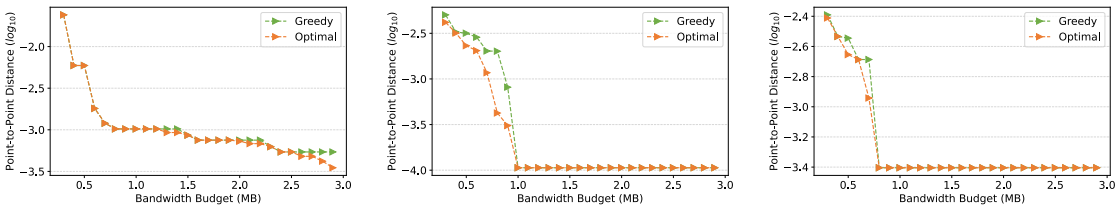


(a) Longdress. (b) Toddler. (c) Andrew.

Figure 4.10: The amount of skipped frames of different streaming schemes in the live volumetric video scenario.

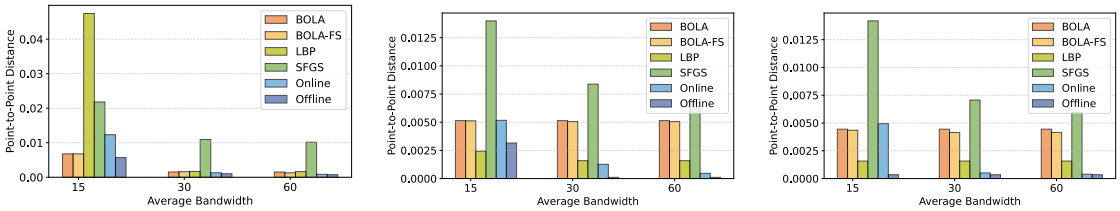
frames. Due to the high number of skipped frames in live volumetric video streaming scenarios, the average video quality closely correlates with the number of skipped frames. Generally, the performance of all schemes improves with increasing average bandwidth. Interestingly, some schemes such as SQLS and SFGS perform better in BW2 (30 MB) than in BW3 (60 MB). This improvement is likely due to the higher fluctuations observed in BW3. Among all schemes, NAS exhibits the poorest performance because it transmits video at a constant quality level, failing to adapt to fluctuating network conditions. NLAS shows improvements over NAS by adapting video quality based on available bandwidth, though it does not perform as well as layer-based approaches. SQLS notably surpasses NLAS in the sequence *longdress*, especially under poor network conditions. However, it performs worse than NLAS for the sequence *toddler* and *andrew*. It suggests that both layering and encoding are crucial for adaptive volumetric video streaming. SFGS outperforms all baseline schemes under different network conditions and sequences. Its flexibility in selecting and combining different layers makes it particularly effective for variable network conditions.

To further assess the performance of SFGS, we compare it with the optimal solution, as shown in Fig. 4.11. We use the logarithmic scale to emphasize the differences in point-to-point distances. In *longdress*, SFGS achieves optimal performance under a small bandwidth budget since it selects a single layer in this scenario. As the bandwidth



(a) Longdress. (b) Toddler. (c) Andrew.

Figure 4.11: The amount of skipped frames of different streaming schemes in the live volumetric video scenario.



(a) Longdress. (b) Toddler. (c) Andrew.

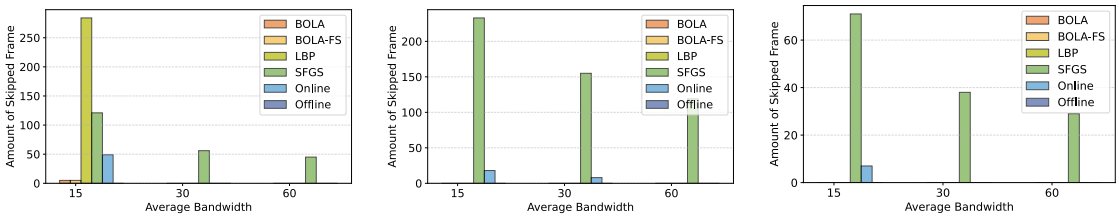
Figure 4.12: The average video quality of different prefetching schemes in the on-demand volumetric video scenario.

increases, SFGS achieves near-optimal performance. However, for sparser volumetric videos such as *toddler* and *andrew*, a gap exists between SFGS and the optimal solution under small bandwidth budgets because the greedy algorithm selects a single layer with a high quantization parameter, while the optimal solution combines two layers with lower quantization parameters. As the bandwidth budget exceeds a threshold, SFGS achieves the optimal performance because it selects all layers with the highest quantization parameter, aligning with the optimal solution.

4.3.4.4 Performance in On-Demand Scenarios

In this subsection, we evaluate and compare the performance of various prefetching schemes in on-demand volumetric video streaming scenarios. To further highlight the advantages of prefetching, we also include the SFGS algorithm as a baseline.

The average video quality and skipped frame amount of different prefetching schemes



(a) Longdress. (b) Toddler. (c) Andrew.

Figure 4.13: The amount of skipped frames of different prefetching schemes in the on-demand volumetric video scenario.

are illustrated in Fig. 4.12 and Fig. 4.13. Compared with SFGS, prefetching techniques improve performance by fetching future frames in advance. Specifically, BOLA-based prefetching schemes effectively reduce skipped frames, thereby improving average video quality. Though BOLA-FS allows quality upgrade of transmitted frames, its benefit is limited due to the high bandwidth consumption required to retransmit high-quality versions. In contrast, LBP leverages layer partitioning to enable progressive quality enhancement, achieving strong performance when bandwidth is not severely constrained. However, under BW1 (15 MB), LBP performs poorly for the *longdress* sequence, even worse than SFGS, as *longdress* has a larger size compared to the other sequences. Hence, the bandwidth required to transmit a single layer with $qp = 9$ exceeds the available throughput, resulting in a high number of skipped frames. The proposed layer-based prefetching schemes address this issue by introducing variable quantization parameters to support diverse videos and network conditions. The offline algorithm always performs best except for *toddler* under BW1 (15 MB), where LBP achieves the best performance. Such a gap occurs because the relationship between frame quality and bandwidth in real videos does not strictly follow a concave function, limiting the optimization potential of the offline algorithm. The online algorithm achieves a good performance under good network conditions (BW2 and BW3), but it still results in skipped frames and low video quality due to limited information on future bandwidth conditions.

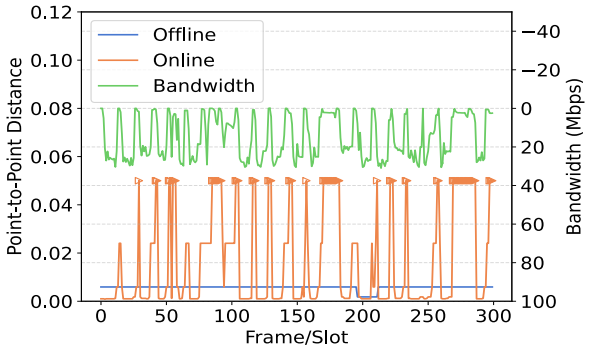


Figure 4.14: Comparison between online and offline layer-based prefetching.

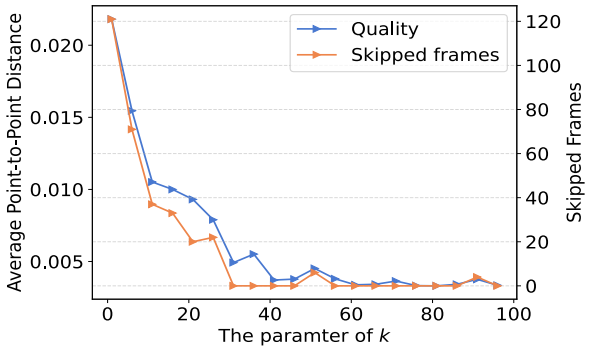


Figure 4.15: Impact of k for the online prefetching-enabled streaming algorithm.

To further analyze the online and offline algorithms, we compare their performances (for *longdress* with BW1), as shown in Fig. 4.14. Compared with the offline algorithm, the online algorithm skips more frames and has more quality switches due to the lack of future bandwidth information. Fig. 4.15 explores the impact of the length of the prediction window k on video quality and skipped frames. Generally, with the increase of k, the video quality increases and the number of skipped frames decreases because the algorithm can utilize more information about future bandwidth conditions and achieve better prefetching results. However, when k is greater than a threshold, the improvement becomes insignificant.

4.3.4.5 A Running Example

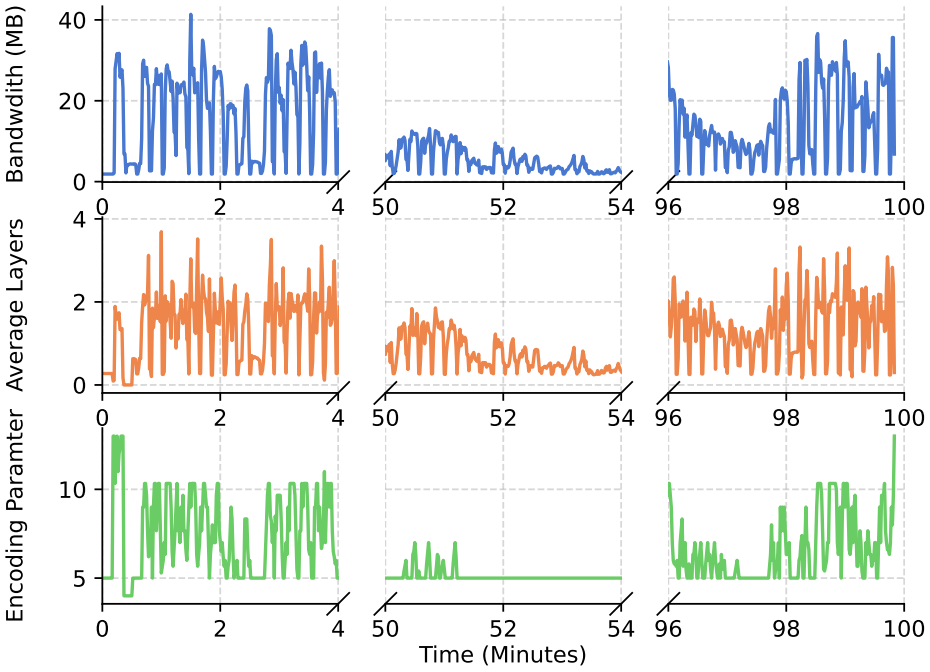


Figure 4.16: A running example.

In this subsection, we provide a running example under highly variable network conditions to show the run-time behavior and assess the robustness of PLVS in the frame-skip scenario, as illustrated in Fig.4.16. We create a long video sequence by playing the *longdress* sequence 100 times and stream it to the clients under real-world network conditions. The bandwidth trace exhibits significant variability throughout the duration such as sudden disruptions. The PLVS streaming framework dynamically adjusts the number of layers and their encoding parameters to optimize video quality under dynamic network conditions. For instance, during periods of high bandwidth, such as between minutes 98 and 100, the scheme downloads an average of three layers per slot with high encoding parameters. This strategy enhances the video quality, ensuring that viewers receive the best possible visual experience during these optimal conditions. Conversely, when the bandwidth diminishes, as observed between minutes 52 and 54, the streaming scheme responds by reducing the number of downloaded layers and selecting lower

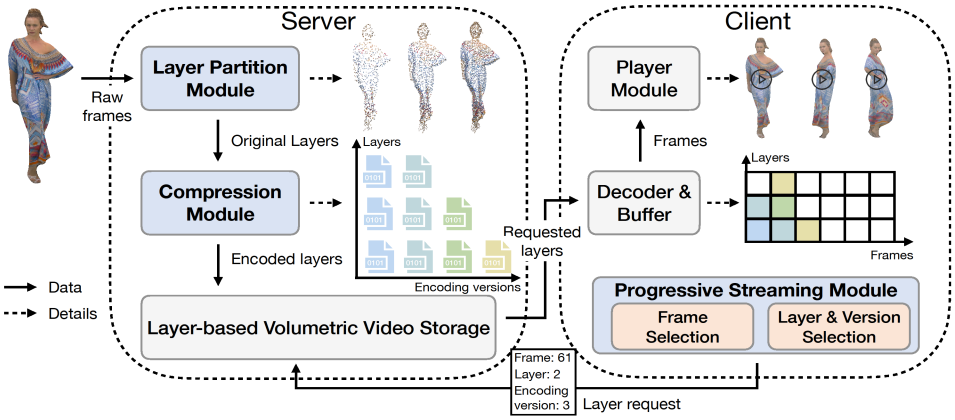


Figure 4.17: System overview of PLVS for the non-frame-skip scenario.

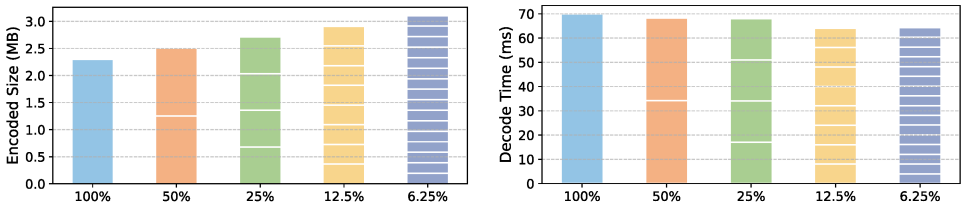
encoding parameters. This adjustment is crucial to decrease the number of skipped frames, even though it may compromise the video quality temporarily.

4.4 PLVS Design for the Non-Frame-Skip Scenario

4.4.1 System Overview

The progressive layer-based volumetric video streaming system aims to improve QoE under dynamic networks by providing quality refinement. The system is composed of a server for volumetric video storage and streaming in layered format, and a client for making adaptive streaming decisions and rendering the video content for display, as shown in Fig.4.17.

On the server side, the raw volumetric video frames are first processed by the **Layer partitioning Module**, which divides each frame into multiple layers. Each layer contains a subset of points of the frame, representing different levels of detail. These layers are then passed to the **Compression Module** for encoding. Since not all encoding versions are suitable for each layer (detailed in Section 4.4.3), the compression module only encodes layers with suitable versions. Afterwards, the compressed layers are stored



(a) Encoded size of layers with different point density.

(b) Decode time of layers with different point density.

Figure 4.18: Overhead brought by layer partitioning.

in the server to support progressive transmission according to client requirements.

On the client side, the **Progressive Streaming Module** is responsible for determining the frames, layers, and encoding versions to request from the server. Based on current network conditions and buffer levels, it sends requests to the server accordingly. The transmitted data is stored in the buffer until they are ready for playback. The Player retrieves frames from the buffer, assembles layers, and renders them for display.

By leveraging this layered structure, the system is able to deliver volumetric video progressively, hence improving the quality of experience even under dynamic network conditions. The design details are presented in the following subsections.

4.4.2 Layer Partitioning Module

The **Layer Partitioning Module** is designed to divide raw volumetric video frames into multiple layers. In point cloud-based volumetric videos, each layer represents a subset of points. The combination of different layers allows progressive quality refinement of videos. Given this purpose, the key challenge lies in determining an effective layer partitioning strategy.

A uniform partitioning strategy is a straightforward but effective approach, where volumetric video frames are uniformly sampled into subsets of points. While this ap-

proach provides flexibility for progressive transmission, it introduces data size overhead. That is, layer-based transmission results in a larger total data size than transmitting the original frame as a whole. Fig. 4.18 illustrates this overhead with layers of varying point densities. In Fig. 4.18(a), as the point cloud density of a layer decreases, the data size overhead increases. For example, transmitting 16 layers with 6.25% of points each results in a data size 35% larger than that of the 100% one. Fortunately, as shown in Fig. 4.18(b), layer partitioning does not bring extra decoding overhead. In fact, it slightly reduces decoding time due to less efficient compression at lower densities.

To reduce data size overhead, we employ a binary tree partitioning structure. In this structure, the original frame, as the root node, contains all points, and layers in the successive depth have half of the points of their parent layer. Specifically, the original frame at the root contains 100% points, while the first depth layer contains 50%, the second depth layer contains 25%. This module includes up to 5 depths, resulting in point cloud densities ranging from 6.25% to 100%. This approach allows any two sibling layers within the same depth to be combined to reconstruct their parent layer, and layers across all depths can be merged to recover the complete point cloud of the original frame.

The binary-tree partitioning approach mitigates the data size overhead by providing various densities. For example, instead of streaming four 6.25% layers to reach a 25% point density, we can directly transmit a 25% layer, effectively decreasing the data size from 0.78 MB to 0.68 MB, hence optimizing transmission efficiency. This partitioning strategy offers more layers and is well-suited for progressive streaming in dynamic network conditions.

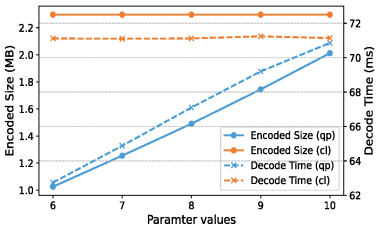


Figure 4.19: Encoded size and decoding time for different encoding parameters.

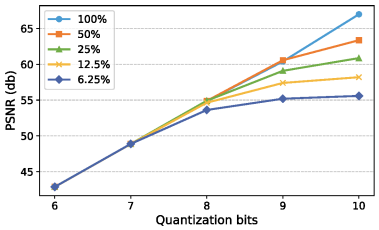


Figure 4.20: Quality of various layers with different encoding versions.

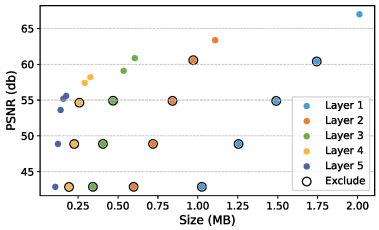


Figure 4.21: Quality and size of various layers with different encoding versions.

4.4.3 Layer-aware Compression Module

Compression is one of the most important components in volumetric video streaming, since raw data is too large to transmit directly. Google Draco [4] is one of the most popular codecs because it supports real-time volumetric video decoding [27, 125]. To further support layer flexibility, multiple encoding versions are needed for compression. Google Draco includes three primary components: quantization, k-d tree compression, and entropy compression. The quantization module uses a quantization bits parameter (qp) to control the precision of position information, while the compression level (cl) specifies the level of compression applied in the k-d tree and entropy compression modules. Here, cl = 0 means no k-d tree compression, while other values represent different combinations of entropy compression algorithms.

To define the encoding versions in this module, we conduct experiments to measure the impacts of these parameters, as shown in Fig. 4.19. The results indicate that the quantization bits parameter (qp) has the most significant influence on both compression size and decode time, as it directly alters the content being compressed. In contrast, k-d tree and entropy compression are lossless methods that have little impact on compression size and decode time. Furthermore, qp introduces lossy compression, leading to variations in video quality. Therefore, we use different quantization bits (qp) to define the encoding versions, achieving a balance among data size, decode time, and video quality.

Moreover, we observed that at low quantization parameters (e.g., 6, 7 bits), layers with different point densities yield similar visual quality. In other words, increasing point density under these settings does not lead to a noticeable improvement in quality but results in larger compression sizes and longer decoding times, as shown in Fig. 4.20. This observation indicates that if we naively apply the same encoding parameters across all layers, it would lead to unnecessary data transmission and increased processing time without quality gains.

To address this inefficiency, we developed the Layer-aware Compression Module, which adjusts quantization parameters based on each layer,Äôs point density. As shown in Fig. 4.21, we exclude combinations of layers and encoding versions that result in low quality but large data sizes. Consequently, layers with the lowest quality are encoded using the full range of quantization bit parameters, while higher-quality layers are limited to higher quantization settings, as lower settings fail to improve quality but increase compression size and decoding time. This layer-aware adjustment optimizes compression efficiency by avoiding unnecessary encoding.

With the **Layer Partitioning Module** and **Layer-aware Compression Module**, the raw video data are segmented into layers and compressed into different encoding versions. These encoded frames are saved in the server and transmitted to the clients as required, enabling progressive streaming.

4.4.4 Progressive Streaming Module

The **Progressive Streaming Module** in the client adaptively requests frames, layers, and encoding versions based on current network conditions and buffer levels, to maximize user QoE during progressive streaming. To determine the optimal streaming strategy, we first formulate a QoE optimization problem and then break it down into two

Table 4.5: Table of notations for PLVS for the non-frame-skip scenario

Variable	Description
$\mathcal{F} = \{f_1, ..., f_N\}$	Set of frames in the volumetric video
$\mathcal{L}_i = \{l_{i,1}, ..., l_{i,m}\}$	Set of layers for frame f_i
$\mathcal{V}_{i,j} = \{v_{i,j,1}, ..., v_{i,j,K_{i,j}}\}$	Encoding versions for layer $l_{i,j}$
$\mathcal{R}_i$	Set of received data chunks for frame f_i
$\mathcal{R}_i^*$	Selected subset of $\mathcal{R}_i$ that maximizes quality
Q^{video}	Average video quality over all frames
I^{switch}	Quality switching impairment across frames
I^{stall}	Total playback stall over all frames
t_i^{expected}	Expected playback time of frame f_i
t_i^{actual}	Actual playback time of frame f_i
$t_i^{\text{request}}(\cdot)$	Time when data is requested
$\tau^{\text{down}}(\cdot)$	Data download duration
$\tau^{\text{decode}}(\cdot)$	Data decode duration
$t^{\text{buffer}}(\cdot)$	Time when data becomes available in buffer
τ	Duration of a single frame (playback duration)
B_t	Current buffer level at time t
τ_t	Transmission duration allocated at time t
$\hat{c}_t$	Bandwidth prediction at time t
S_t	Data size allowed for transmission at time t
$\phi_i(S_t)$	Mapping from budget S_t to optimal data for frame f_i

sub-problems. We introduce two key components: *Layer and Version Selection* and *Frame Selection*, each responsible for a specific sub-problem. The notations used in the problem formulation and algorithms are listed in Table 4.5

4.4.4.1 QoE optimization problem formulation

As most works [114, 121, 128] do, we define QoE as an overall measurement composed of **video quality** Q^{video}, **video quality switch impairment** I^{switch} and **video stall impairment** I^{stall}.

The **video quality** Q^{video} is defined as the average frame quality Q^{frame}. Assuming there are N frames of a volumetric video $\mathcal{F} = \{f_1, f_2, \cdots, f_N\}$, then the video quality is

calculated as,

$$Q^{\text{video}} = \frac{1}{N}\sum_{i=1}^{N} Q_i^{\text{frame}}. \tag{4.17}$$

The frame quality Q_i^{frame} is determined by the received data of the frame f_i. There are m layers $\mathcal{L}_i = \{l_{i,1}, l_{i,2}, \cdots, l_{i,m}\}$ of the frame f_i, and each layer $l_{i,j}$ has $K_{i,j}$ encoding version $\mathcal{V}_{i,j} = \{v_{i,j,1}, v_{i,j,2}, \cdots\}$. Given the received data $\mathcal{R}_i = \{r_{i,1}, r_{i,2}, \cdots\}$, where $r_{i,j} = (l_{i,j_1}, v_{i,j_1,k_1})$, we have observed that simply combining all data in $\mathcal{R}_i$ may not improve frame quality as expected. It is because the layer with a high-compression version brings too high compression loss which can not be compensated by others. In this example, the quality of the combination of a 12.5%-layer with qp=8 and a 12.5%-layer with qp=6 is lower than a 12.5%-layer with qp=8 alone. Hence, it is essential to select an optimal subset of received data $\mathcal{R}_i^*$ to optimize frame quality,

$$\mathcal{R}_i^* = \max_{\mathcal{R}' \in 2^{\mathcal{R}_i}} Q(\mathcal{R}'), \tag{4.18}$$

$$Q_i^{\text{frame}} = Q(\mathcal{R}_i^*). \tag{4.19}$$

The **video quality switch impairment** I^{switch} is the average quality switch impairment between successive frames I_i^{switch}, which is measured by the quality difference between the current frame i and the previous frame $i-1$,

$$I^{\text{switch}} = \frac{1}{N}\sum_{i=2}^{N} I_i^{\text{switch}} = \frac{1}{N}\sum_{i=2}^{N} |Q_i^{\text{frame}} - Q_{i-1}^{\text{frame}}|. \tag{4.20}$$

The **video stall impairment** I^{stall} is the sum of all single frame stall I_i^{stall},

$$I^{\text{stall}} = \frac{1}{N}\sum_{i=1}^{N} I_i^{\text{stall}}. \tag{4.21}$$

The frame stall I_i^{stall} is the difference between its expected play time t_i^{expected} and its actual play time t_i^{actual},

$$I_i^{\text{stall}} = t_i^{\text{actual}} - t_i^{\text{expected}}. \tag{4.22}$$

The expected play time t_i^{expected} is the sum of the frame play duration τ and the expected play time of the previous frame $t_{i-1}^{\text{expected}}$,

$$t_i^{\text{expected}} = \tau + t_{i-1}^{\text{expected}}. \tag{4.23}$$

The actual play time t_i^{actual} is the bigger value between the time when the first received data $r_{i,1}$ of the frame f_i are saved to the buffer $t^{\text{buffer}}(r_{i,1})$ and the expected play time t_i^{expected},

$$t_i^{\text{actual}} = \max\{t^{\text{buffer}}(r_{i,1}), t_i^{\text{expected}}\}. \tag{4.24}$$

The time when the first downloaded data of the frame $r_{i,1}$ saved to the buffer $t^{\text{buffer}}(r_{i,1})$ is the sum of the request time $t_i^{\text{request}}(r_{i,1})$, the download duration $\tau_i^{\text{down}}(r_{i,1})$ and the decode duration $\tau_i^{\text{decode}}(r_{i,1})$,

$$t^{\text{buffer}}(r_{i,1}) = t_i^{\text{request}}(r_{i,1}) + \tau^{\text{down}}(r_{i,1}) + \tau_i^{\text{decode}}(r_{i,1}), \tag{4.25}$$

where download duration is the ratio between the layer size $s(r_{i,1})$ and current bandwidth $c(t)$, $\tau^{\text{down}}(r_{i,1}) = \frac{s(r_{i,1})}{c(t)}$.

Hence, the final **QoE model** is the weighted sum of all these three components,

$$QoE = \omega_1 Q^{video} - \omega_2 I^{\text{switch}} - \omega_3 I^{\text{stall}} \tag{4.26}$$

where ω_1, ω_2 and ω_3 are weighted parameters.

The QoE maximization problem aims to decide the data to transmit $\mathcal{D} = \{d_1, d_2, \cdots d_t, \cdots\}$, where $d_t = (f_{i_t}, l_{i_t,j_t}, v_{i_t,j_t,k_t})$, to optimal QoE:

$$\max_{\mathcal{D}} QoE = \omega_1 Q^{video} - \omega_2 I^{\text{switch}} - \omega_3 I^{\text{stall}} \tag{4.27}$$

$$\text{s.t.} \quad \begin{aligned} & f_{i_t} \in \mathcal{F}, l_{i_t,j_t} \in \mathcal{L}_{i_t}, v_{i_t,j_t,k_t} \in \mathcal{V}_{i_t,j_t}, \\ & \forall (f_{i_t}, l_{i_t,j_t}, v_{i_t,j_t,k_t}) \in \mathcal{D} \end{aligned} \tag{4.28a}$$

$$\mathcal{R}_i = \{d_t \mid f_{i_t} = f_i, t^{\text{buffer}}(d_t) \le t_i^{\text{actual}}\}. \tag{4.28b}$$

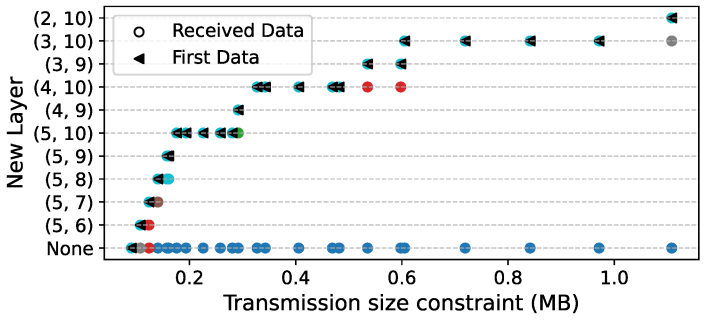


Figure 4.22: Optimal layer selection for received data.

Constraint (4.28a) specifies the valid frames, layers, and encoding versions available for transmission, while constraint (4.28b) defines the layers of each frame that have been received by the time it is going to play.

In the non-frame-skip scenario, the primary trade-off involves maintaining high video quality versus minimizing playback stalls. Delivering high-quality streams increases the risk of rebuffering, whereas lowering quality helps prevent stalls but may degrade the viewing experience.

It is difficult to solve the QoE maximization problem directly because of complex decision spaces. To address this, we decompose the problem into two sub-problems: the layer and encoding version selection, and the frame selection. In the first sub-problem, we aim to determine the optimal combination of received layers with various encoding versions. In the second sub-problem, we address the trade-off between enhancing the quality of previously received frames by refining old frames and maintaining playback smoothness by transmitting new frames.

4.4.4.2 Layer and Encoding Version Selection

The above observation that additional data does not always improve quality highlights the need for a layer selection strategy to avoid such potential transmission wastage.

The most straightforward approach is to perform a brute-force search to determine the optimal layer combination given a transmission size constraint. However, this method is impractical due to the vast search space. Instead of finding the final combination directly, we use a greedy algorithm to select a single optimal layer based on the current transmission size constraint and received data.

We conduct an offline brute-force search to establish a mapping from specific transmission sizes and received data sets to their corresponding optimal layer selections. Since the transmission data size is continuous and hence infinite, we approximate it by using the sizes of individual layers. The Fig. 4.22 presents the new layer selection based on received data and transmission size constraint. The "None" data in this figure indicates that no additional layer would improve quality, as previously discussed. The results demonstrate that the optimal layer selection with received data closely matches the optimal choice for the first new data. Therefore, we use this mapping $\phi_i : \mathbb{R}^+ \rightarrow \{(l_{i,j}, v_{i,j,k}) | l_{i,j} \in \mathcal{L}_i, v_{i,j,k} \in \mathcal{V}_{i,j}\}$ from the transmission data size $S_t \in \mathbb{R}^+$ to the optimal data selection $(l^*_{i,j}, v^*_{i,j,k})$ to guide the layer and encoding selection for the frame f_i. If the new data does not improve the current quality given the current received data $\mathcal{R}_{i,t}$, the transmission will be skipped.

4.4.4.3 Frame selection

To provide the transmission data size constraint S_t and current received layers $\mathcal{R}_{i,t}$ for the mapping ϕ_i, we need to determine the duration τ_t allocated for the current transmission and the frame f_i to download. Given the uncertainty in bandwidth prediction, we adopt buffer-based algorithms to dynamically adjust transmission duration τ_t and frame f_i according to current buffer levels, thereby improving resilience to bandwidth fluctuations.

The **Buffer-based Transmission Duration Algorithm** is based on the classical

Algorithm 4.5 Buffer-based Transmission Duration Algorithm

Input: Current buffer state B_t, reservoir buffer size B^{res}, cushion buffer size B^{cus}, minimum duration τ^{min}, maximum duration τ^{max}, adjustment factor α

Output: Allocated transmission duration τ_t

1 **if** $B_t \leq B^{\text{res}}$ **then**
2 $\quad \tau_t \leftarrow \tau^{\min}$
3 **else if** $B_t \geq B^{\text{cus}}$ **then**
4 $\quad \tau_t \leftarrow \tau^{\max}$
5 **else**
6 $\quad$ Calculate the download duration rate factor $F \leftarrow \frac{B_t - B^{\text{res}}}{B^{\text{cus}}}$
7 $\quad$ Calculate the actual download duration $\tau_t \leftarrow \tau^{\min} + F \times (\tau^{\max} - \tau^{\min})$
8 $\quad$ Adjust the download duration $\tau_t \leftarrow \tau_t \times \alpha$
9 **end**
10 **return** τ_t

Buffer-based Approach (BBA) [39], as illustrated in Alg. 4.5. There are three stages in this algorithm. When the buffer level is below the reservoir threshold B^{res}, it adopts an aggressive strategy by setting the minimum transmission duration τ^{min} to quickly download more frames (lines 1-2). Conversely, when the buffer exceeds the cushion threshold B^{cus}, it switches to a conservative mode with maximum transmission duration τ^{max} (lines 3-4) to prioritize video quality. For buffer levels between these thresholds, a rate factor F is calculated through linear mapping (line 6) and used to determine the transmission duration (line 7). An additional adjustment factor α fine-tunes the transmission duration τ_t (line 8).

We integrate frame, layer, and encoding version selection into the **Buffer-based Progressive Streaming Algorithm**. It first selects old frames for enhancement or the new frame for playback smoothness and then determines the corresponding layer and encoding version using the mapping ϕ_i, as illustrated in Alg. 4.6. Initially, it constructs a candidate set of frames in the buffer that can be played after the transmission duration (line 1). When the buffer level falls below the reservoir threshold B^{res}, it prioritizes playback smoothness by downloading the newest frame (lines 3-4). Otherwise, the

Algorithm 4.6 Buffer-based Frame Selection Algorithm

Input: Current buffer level B_t, reservoir buffer size B^{res}, allocated transmission duration τ_t, received layers in the buffer $\mathcal{R}^{\text{buffer}}$, current time t, current bandwidth prediction $\hat{c}_t$

Output: The frame to download f_i

1 Initialize the set as frames in the buffer with enough time to play $\mathcal{F} \leftarrow \{f_i \mid (f_i, l_{i,j}, v_{i,j,k}) \in \mathcal{R}^{\text{buffer}}, t + \tau_t \geq t_i^{\text{actual}}\}$
2 Get the index of the newest frame index i^{new}
3 **if** $B_t \leq B^{res}$ **then**
4 | **return** The new frame f_i^{new}
5 **end**
6 **foreach** *frame* $f_i \in \mathcal{F}$ **do**
7 | Calculate the current quality of frame f_i Q_i^{frame}
8 **end**
9 Sort frame candidates in increasing order of frame quality $\mathcal{F}^{\text{sort}} \leftarrow \text{sorted}(\mathcal{F})$
10 Calculate the transmission data size constraint $S_t \leftarrow \hat{c}_t \times \tau_t$
11 **foreach** *frame* $f_i \in \mathcal{F}$ **do**
12 | Get data to download $r_t \leftarrow \phi_i(S_t)$
13 | **if** r_t *is not None and* $Q(\mathcal{R}_{i,t}) \leq Q(\mathcal{R}_{i,t} \cup \{r_t\})$ **then**
14 | | **return** The frame to download f_i
15 | **end**
16 **end**
17 **return** The new frame f_i^{new}

algorithm evaluates and sorts candidates by their current quality (lines 6-9). Meanwhile, the transmission size constraint is calculated based on predicted bandwidth and allocated duration (line 10). For each candidate f_i, it uses the mapping ϕ_i to identify a feasible combination of layer and encoding version for download. If no combination improves frame quality, the algorithm defaults to downloading the newest frame (line 17). This approach effectively balances the quality enhancement of existing frames with the timely download of new frames, while maintaining buffer stability.

4.4.5 Evaluation and Implementation

In this section, we evaluate the performance of PLVS through extensive simulations and compare it with baselines to demonstrate its improvements in QoE and reductions

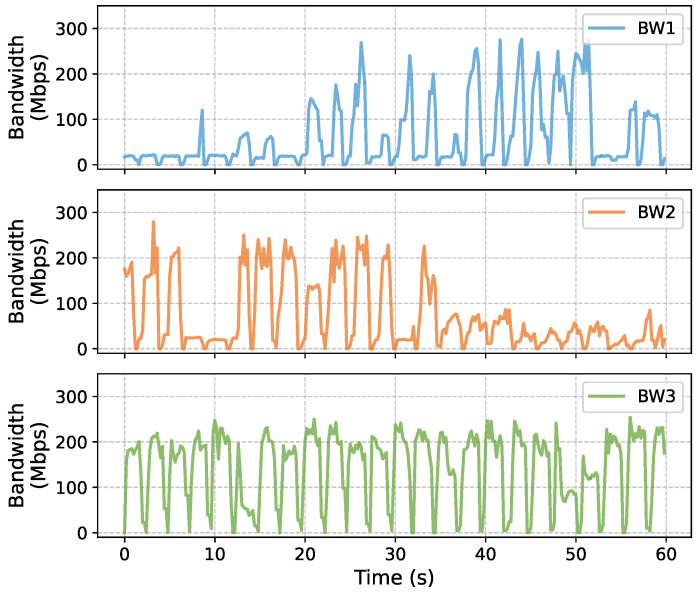


Figure 4.23: Visualization of three network traces.

in bandwidth wastage. Additionally, we implement a prototype system for a detailed case study (Section 4.4.5.3) to show the practical feasibility and effectiveness of PLVS in real-world applications.

4.4.5.1 Experiment Setup

Dataset. We use four volumetric videos from 8i Voxelized Full Bodies (8iVFB vox10) [20]: *Longdress*, *Loot*, *Redand black*, and *Soldier*. Each sequence contains 300 frames. Each video is partitioned into layers with a binary tree structure with a maximum depth of 5, resulting in a minimum point cloud density of 6.25%. These layers are encoded using Google Draco with the default compression level of 7 and five different quantization parameters, $v \in \{6, 7, 8, 9, 10\}$.

Communication Resource. In simulations, we select three network traces from a 5G Dataset [80], as Fig. 4.23 shows. The first trace has an average bandwidth of 64 Mbps and exhibits a low-high pattern. The second trace has an average bandwidth of 70 Mbps

with high throughput in the initial period and low throughput in the later time. The third trace has an average bandwidth of 142 Mbps and has a stable and periodic pattern.

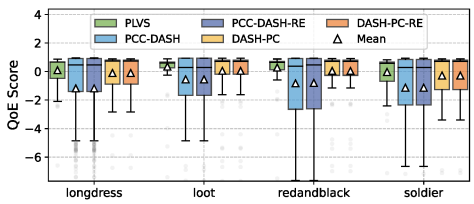
Evaluation Metrics and Algorithm Settings. We measure the volumetric video quality by the Point Cloud PSNR [58, 100] between the encoded video and the original video. The PSNR values are normalized using min-max scaling to the range [0,1], where the minimum corresponds to the PSNR of a single 6.25% layer with qp=6, and the maximum corresponds to the PSNR of the original frame with qp=10. The buffer size is set to 60 frames, with the reservoir buffer size B^{res} and cushion buffer size B^{cus} set to $\frac{1}{4}$ and $\frac{3}{4}$ of the total buffer size, respectively. The adjustment factor α is set to 0.9. For bandwidth prediction, we use a simple heuristic method by assuming the next time slot‚Äôs bandwidth equals the current bandwidth.

Baselines. We compare PLVS with non-progressive volumetric video systems as follows:

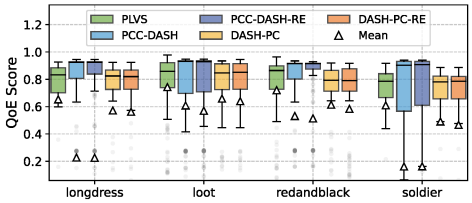
- **PCC-DASH [102]** adaptively selects different encoding parameters (i.e., quantization bits) for volumetric videos according to available bandwidth. For a fair comparison, we use the same set of encoding versions as PLVS. Since the 8i dataset contains a single object per scene, we select the greedy bit rate allocation strategy for this baseline. Unlike PLVS, PCC-DASH does not support quality refinement.

- **PCC-DASH with refinement (PCC-DASH-RE)** is a variant of PCC-DASH with additional refinement scheme. It is enabled only when the buffer state exceeds the reservoir threshold B^{res}. It requests data that maximally improves quality within the available download budget. Due to the high bandwidth requirements of layers with high point cloud density, the layer density is restricted to 25%.

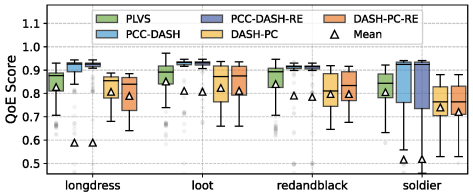
- **DASH-PC [33]** provides multiple point cloud densities, as layers in PLVS. In the



(a) QoE comparison under BW1.



(b) QoE comparison under BW2.



(c) QoE comparison under BW3.

Figure 4.24: QoE performance comparison of different videos under different network traces.

original implementation, there is no adaptive algorithm available. Therefore, we employ the same adaptive algorithm as PCC-DASH to select appropriate layers dynamically. DASH-PC does not reuse previously transmitted layers for quality refinement.

- **DASH-PC with refinement (DASH-PC-RE)** enhances previous frame quality in DASH-PC when bandwidth improves. The refinement strategy follows the same logic as PLVS, but it uses the layer with maximum density without improving the encoding version. To ensure quality improvement by various point cloud densities, we set the encoding version to qp=9 for both DASH-PC-based algorithms.

4.4.5.2 Experimental Results

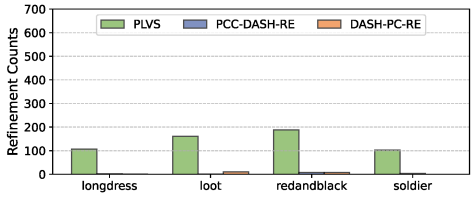
In this subsection, we compare the performance of PLVS with the above baselines under different network conditions. Metrics including QoE, refinement counts, and bandwidth wastage are taken into comparison for four different videos.

Overall QoE Performance. In Fig. 4.24, we compare QoE values of four algorithms

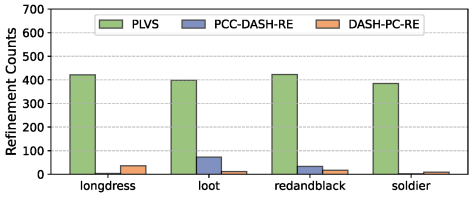
under three traces. Among these, PLVS consistently achieves the highest and most stable average QoE across all scenarios. Its progressive streaming scheme effectively mitigates the impact of network dynamics by allowing incremental quality refinement. PCC-DASH and PCC-DASH-RE have the most unstable performance under low throughput (e.g., BW1) and videos with high data amount (e.g., *soldier*). This instability occurs because the quality gap between different encoding versions is much greater than the data size. Therefore, the video quality varies from frame to frame. DASH-PC and DASH-PC-RE demonstrate more stable QoE compared to PCC-DASH-based schemes because point cloud density variations have a lower quality gap but a higher size gap, as indicated in Section 4.4.3. However, their performance remains lower than PLVS, as they lack encoding version flexibility.

Impact of Video Contents. The performance of all algorithms varies depending on the video content. For instance, in videos with higher data amounts such as *longdress* and *soldier*, QoE values of all algorithms degrade compared to the other two videos, especially for PCC-DASH and PCC-DASH-RE. It indicates that encoding versions alone are not able to handle a great transmission amount. Meanwhile, these videos exhibit the largest gaps in QoE between PLVS and the baselines, because PLVS provides more combinations of layers and encoding versions to reduce QoE degradation.

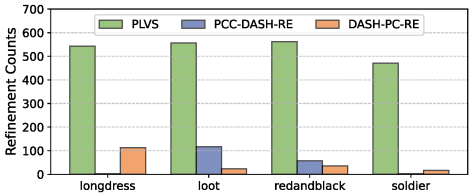
Impact of Bandwidth Traces. The three typical bandwidth traces show varying impacts on QoE values. BW1 trace is of high fluctuation and has a low-high pattern. In this situation, PLVS buffers frames during the low throughput, and gradually improves their quality by transmitting additional layers. However, baselines without refinement can not take advantage of high throughput at a later time, leading to a low QoE. Even refinement-enabled baselines do not significantly improve QoE because low throughput in the early phase keeps the buffer levels low, preventing refinement triggers. On the contrary, BW2 follows a high-low pattern. This steady high-low trend allows all algorithms to perform



(a) Refinement count under BW1.



(b) Refinement count under BW2.

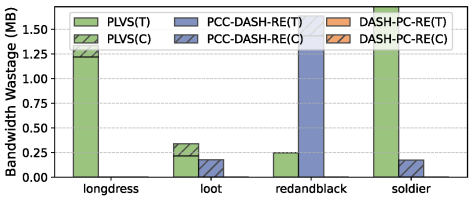


(c) Refinement count under BW3.

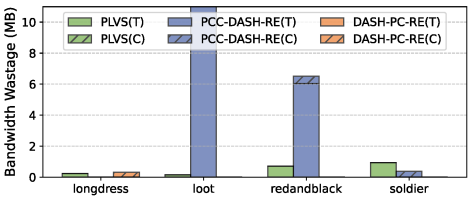
Figure 4.25: Refinement count comparison of different videos under different network traces.

better compared to BW1, despite a similar average throughput. However, refinement-based baselines again fail to leverage their potential because the highest-quality frames are already transmitted during the high-bandwidth period. Once the bandwidth drops, all baselines struggle to transmit new frames. However, PLVS maintains high QoE by efficiently leveraging the initial bandwidth surplus for progressive quality refinement. BW3 provides a stable, periodic bandwidth with a higher average throughput compared to BW1 and BW2. Although this trace yields the best performance for all algorithms, PLVS still outperforms baselines by up to 56% due to progressive quality improvement. Refinement does not always work in this situation because high-quality frames have already been transmitted, leaving little room for refinement.

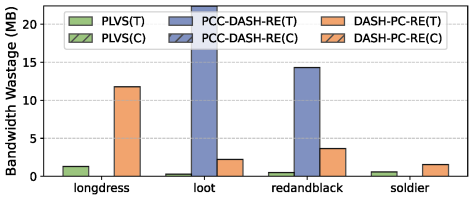
Impact of Refinement. We further analyze the refinement behavior of PLVS compared to PCC-DASH-RE and DASH-PC-RE, as illustrated in Fig. 4.25. The results show that PLVS triggers refinement more frequently than both PCC-DASH-RE and DASH-PC-RE across all bandwidth traces and video sequences. This occurs because the progressive quality refinement approach in PLVS provides more room for improvement.



(a) Bandwidth wastage under BW1.



(b) Bandwidth wastage under BW2.



(c) Bandwidth wastage under BW3.

Figure 4.26: Bandwidth wastage comparison of different videos under different network traces.

In contrast, PCC-DASH-RE and DASH-PC-RE always fail to retransmit due to the lack of further quality improvement opportunities once high-quality frames are downloaded. Refinements increase with average throughput for all algorithms as more bandwidth becomes available to improve quality. Video size also impacts refinement behavior: PCC-DASH-RE performs more refinements for smaller videos like *loot* and *redandblack*, while DASH-PC-RE retransmits more for larger videos like *longdress* and *soldier*.

Bandwidth Wastage. Refinement usually leads to bandwidth wastage and we investigate it in refinement-based algorithms, as illustrated in Fig. 4.26. Bandwidth wastage consists of two types: *transmission wastage (T)*, where transmitted data is downloaded but ultimately not used, and *cancel wastage (C)*, where data being transmitted is canceled because the corresponding frame has already been played. PLVS incurs transmission wastage primarily when low-quality layers (e.g., 6.25% density with qp=6) are requested during low bandwidth or low buffer levels. These low-quality layers may degrade overall video quality when high-quality layers are downloaded later, as discussed in Section 4.4.4. For example, under BW1 where throughput is low and bandwidth fluctuation is high,

PLVS exhibits slightly higher total bandwidth wastage compared to other baselines due to its frequent refinements. However, given the high amount of refinement, the wastage per refinement for PLVS is significantly lower than PCC-DASH-RE and DASH-PC-RE, leading to higher bandwidth efficiency. For PCC-DASH-RE and DASH-PC-RE, the transmission wastage increases with refinement counts. PCC-DASH-RE has higher bandwidth wastage than DASH-PC-RE, indicating that point density is a more suitable option in refinement. Cancel wastage arises from optimistic bandwidth predictions. While it occurs in all three refinement-based algorithms, the wastage remains trivial across all cases due to the small proportion of transmissions interrupted by playback.

4.4.5.3 Implementation and Case Study

We also implement a prototype of PLVS to verify its feasibility. The volumetric video streaming system consists of a Vue front end for display and a Python Flask server for data streaming. The client uses Three.js for 3D rendering and decoding point cloud data with DRACOLoader, leveraging WebGL for GPU-accelerated rendering. User interactions are enabled via OrbitControls, a component of Three.js. On the server side, Flask manages client requests for volumetric video frames stored in a structured directory format, dynamically transmitting frames based on client requests and supporting download cancellation to optimize resource usage.

To further validate the feasibility of the PLVS prototype in practical scenarios, we conduct a case study analyzing runtime streaming behavior and end-to-end latency. In particular, we evaluate the system resilience under real-world network conditions, including sudden drops in available bandwidth and transient disconnections. The system maintains playback continuity through adaptive layer selection and prefetching mechanisms, which help minimize visual disruption. These evaluations demonstrate that PLVS can tolerate moderate network instability and maintain acceptable latency

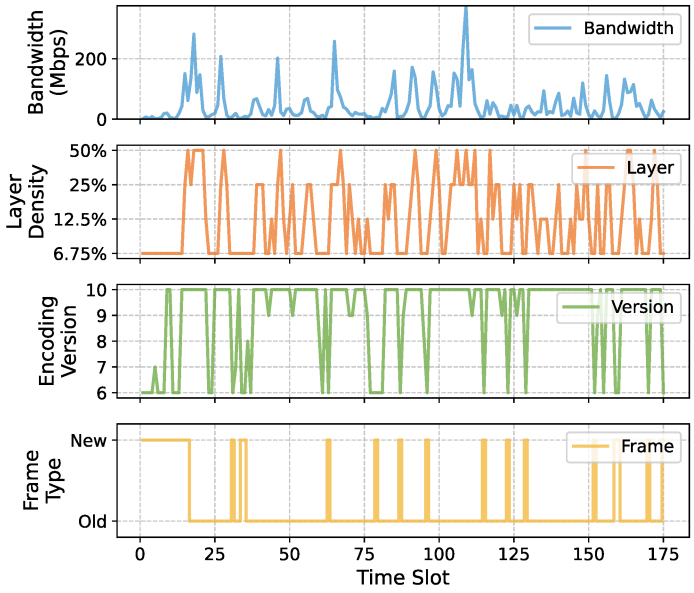


Figure 4.27: Run-time streaming request of PLVS.

and streaming quality, highlighting its potential applicability in real-world deployments with imperfect network conditions.

Case Study. In this case study, we stream the *longdress* video with a low frame rate 15 fps over Wi-Fi using the PLVS prototype. The run-time behaviors are depicted in Fig. 4.27. The layer and encoding version selections dynamically adapt to bandwidth fluctuations, demonstrating PLVS ability to efficiently stream videos under varying network conditions. For instance, during periods of extremely low bandwidth, PLVS selects layers with minimal size and quality (e.g., 6.25% density with qp=6) to ensure uninterrupted playback. As network conditions improve, it progressively switches to higher densities and encoding versions to enhance quality. Additionally, PLVS refines previously streamed frames once the buffer level exceeds a threshold with sufficient bandwidth. For example, during time slots 30-60, PLVS improves older frames by transmitting high-quality layers, such as 50% density with qp=10.

End-to-End Latency Breakdown. We analyze the end-to-end latency breakdown

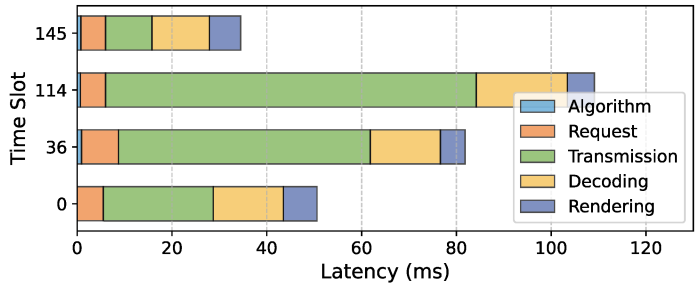


Figure 4.28: End-to-end latency breakdown.

for the above case study, as depicted in Fig. 4.28. The overhead introduced by the progressive streaming algorithm is less than 1 ms, demonstrating its negligible impact on overall latency. Request delay is not included in the simulations, but it lasts around 5 ms depending on the bandwidth condition. Transmission delay is highly dependent on network conditions as well. It accounts for the largest portion of latency, constituting between 28% and 72% of the total. Decoding and rendering times remain stable, with a combined duration of 20 ms. These results highlight PLVS potential to support real-time volumetric video streaming at lower frame rates (e.g., 15 fps, latency < 66 ms). However, even under high bandwidth conditions (e.g., time slot 145), decoding and rendering latencies remain bottlenecks for higher frame rates.

4.5 Chapter Summary

In this chapter, we propose PLVS, a progressive layer-based volumetric video streaming framework designed to address limited and fluctuating bandwidth conditions.

The key contributions are summarized as follows:

- We introduce a layer partitioning strategy that supports progressive quality refinement by reusing previously transmitted data, thereby improving bandwidth efficiency.

- We design and analyze PLVS under two practical scenarios: frame-skip and non-frame-skip, each with distinct optimization goals and algorithmic solutions.
- In the frame-skip scenario, PLVS enhances video quality and reduces skipped frames; in the non-frame-skip scenario, it improves overall QoE and reduces bandwidth wastage.

These results demonstrate the effectiveness of progressive layer-based streaming in adapting to bandwidth variability while maintaining high-quality immersive media experiences.

CHAPTER 5

TILE-BASED VOLUMETRIC VIDEO STREAMING UNDER COMPUTING AND BANDWIDTH CONSTRAINTS

5.1 Introduction

In tile-based volumetric video streaming systems, tile size plays a critical role in determining culling performance, which directly impacts computing and bandwidth resource consumption. Smaller tiles improve culling accuracy by filtering out more redundant data, thereby reducing bandwidth consumption. However, excessively small tiles are impractical as they increase computation overhead due to the larger number of tiles that need processing. Conversely, larger tiles retain more invisible content, leading to higher bandwidth consumption but lower computing overhead. Additionally, smaller tiles degrade compression efficiency because each tile is compressed independently, limiting inter-tile redundancy reduction [27].

Most existing works [27, 54] adopt even 3D tiling and employ a fixed tile size. These approaches usually select a suitable tile size offline and keep it constant during stream-

ing. Authors in [59] propose a new saliency-based tiling scheme. They first partition the volumetric video into fine-grained tiles and cluster them until reaching a distance threshold. However, these approaches don't consider the flexible tile size during volumetric video streaming. Although these methods employ tiling in volumetric video streaming, the fixed tile size limits their performance, especially when facing dynamic and diverse network conditions and user watching behaviors.

We observed that the optimal tile size is highly dynamic, varying significantly with changing network conditions and user viewports. For instance, when the network condition is good enough, computation overhead becomes the bottleneck rather than transmission overhead. In such cases, a larger tile size is preferable, as it reduces computation overhead despite causing some redundant transmission. Conversely, when the network condition becomes worse, a small tile size is necessary to minimize transmission delay and optimize performance. Viewports also play a critical role in tile size selection. For example, a large tile size is preferred when the entire content is visible to reduce computation overhead, while a smaller one is suitable for scenarios where users focus on detailed elements. Therefore, it is essential to dynamically adjust tile size to effectively balance the trade-off between computation and communication overhead.

Additionally, bitrate allocation is a key factor in volumetric video streaming, as it directly determines the visual quality. The unique 3D viewports require different bitrates for each tile within the same frame. For example, tiles at a shorter distance should be allocated a higher density than those at a longer distance because they occupy a larger portion of the viewport and contribute more significantly to the visual quality. Most existing works [58, 75, 105] focus on bitrate allocation for fixed tile sizes. However, the tile size selection influences decisions of tile bitrate allocation. Smaller tile sizes increase the total number of tiles, providing finer granularity for prioritizing bitrate allocation, but also introducing higher computational and transmission overhead. Conversely, larger

tile sizes reduce the number of tiles and simplify bitrate allocation but may compromise spatial importance and reduce compression efficiency. These trade-offs highlight the interaction between tile size selection and bitrate allocation, which requires a new approach to jointly optimize these factors for Quality of Experience (QoE) maximization.

In this chapter, we address the tile configuration problem in volumetric video streaming, including tile size selection and tile bitrate allocation. Instead of using a fixed tile size, we adaptively adjust tile size according to dynamic viewports, video contents, and bandwidths. This adaptive strategy ensures a balance between computational efficiency and transmission overhead. After tile selection, we allocate different bitrates to tiles to optimize the long-term QoE. However, we face two main challenges when designing it.

The most naive way to select a proper tile size is to run culling algorithms with all possible tile sizes. However, this approach is infeasible due to the high computational overhead. Meanwhile, it is challenging to predict culling results in advance. Despite the existence of several viewport prediction algorithms in volumetric videos [38, 66], they do not accurately reflect visible tiles after culling. Due to the vast viewport space and dynamic video contents, it is impractical to establish a direct mapping between predicted viewports and final culling results. However, without information on culling results, it is hard to select a suitable tile size and bitrate accordingly. To address this, reinforcement learning (RL) provides a promising approach by enabling the system to learn optimal decisions from past experience. Unlike traditional methods, RL can make decisions dynamically without requiring explicit knowledge of the culling results. Given that tile size selection and bitrate allocation are sequential decisions, we adopt hierarchical reinforcement learning (HRL) to optimize long-term QoE by tackling these interdependent tasks systematically.

The second challenge arises from the interdependence between tile size selection and

tile bitrate allocation. The tile size directly impacts the culling results, such as culling accuracy and culling time. The tile bitrate allocation is highly dependent on these results. For example, if the culling accuracy is high and most invisible tiles are removed, allowing a higher bitrate to improve visual quality. Moreover, if the culling process takes too long, the reduced transmission time prevents high bitrate allocation. Such interdependence makes it challenging for traditional RL approaches to effectively handle such coupled and sequential tasks, since they are not inherently designed for it. To address this, we adopt an HRL framework [40, 52, 77]. We decouple the tile configuration problem into two stages, the tile size selection (TSS) stage and the tile bitrate allocation (TBA) stage. This decoupling allows each agent to specialize in its task. The TSS agent focuses on tile size selection, while the TBA agent aims at tile bitrate allocation. Furthermore, the HRL framework enables cooperation between two agents, improving the final QoE.

In summary, our contributions are as follows:

- We introduce flexible tile sizes in volumetric video streaming and consider the culling computation overhead. We jointly consider the tile size selection and tile bitrate allocation, and formulate them as an MDP problem to optimize the long-term QoE.
- To solve the MDP problem, we propose a two-stage HRL approach, including a TSS agent and a TBA agent. We employ the Proximal Policy Optimization-based (PPO) for both agents. This HRL framework effectively adapts to dynamic viewports, varying video content, and fluctuating network bandwidth conditions.
- We evaluate the performance of our schemes through extensive trace-driven simulations. We compare our work with existing volumetric video streaming schemes with a fixed tile size. Results indicate that the HRL-based tile configuration approach improves QoE.

5.2 Motivation

This work is motivated by the key observation that tile size significantly influences both culling performance and compression efficiency, which are critical factors in volumetric video streaming. Specifically, tile size affects the granularity of the culling algorithm. It is highly relevant to the accuracy in identifying visible tiles and the computational overhead required for processing. At the same time, tile size impacts compression efficiency, as smaller tiles reduce inter-tile redundancy but increase the number of tiles to encode and transmit. These trade-offs have a direct impact on the overall streaming performance, especially under dynamic conditions such as varying user viewports and bandwidth conditions.

To better illustrate these effects, we first conduct experiments to visualize how tile size affects culling performance and compression efficiency. These experiments highlight the importance of tile size in volumetric video streaming. Furthermore, we provide a toy example to demonstrate the advantages of using flexible tile sizes in volumetric video streaming. This example shows that selecting a suitable tile size under varying conditions can optimize resource utilization and enhance QoE. Together, these findings motivate the development of a volumetric video streaming framework with flexible tile sizes.

5.2.1 Impact of Tile Size on Culling and Compression Performance

Volumetric Video and Viewport Dataset. We conduct measurements and analysis on a representative volumetric video *longdress* from the 8i Voxelized Full Bodies version 2 (8iVFB v2) [20]. The video has an original spatial resolution of 1024×1024×1024. To evaluate the impact of tile size, we divide the video into three tile configurations

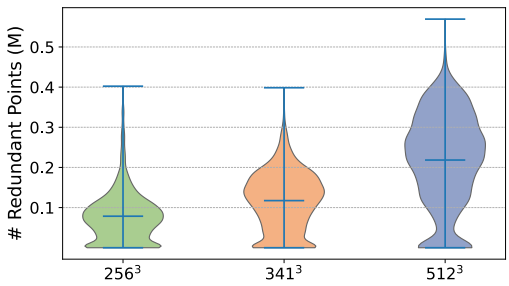


Figure 5.1: Culling accuracy of different tile sizes.

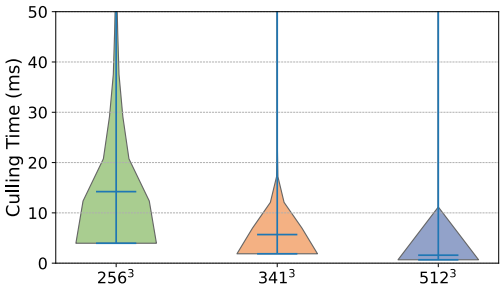


Figure 5.2: Culling time of different tile sizes.

with sizes of 512×512×512, 341×341×341, and 256×256×256. To simulate diverse user viewing patterns, we employ a viewport dataset [21] that provides 40 unique user viewport trajectories for the video *longdress*. This dataset ensures comprehensive coverage of different user behaviors, including scenarios with varying focus areas and motion dynamics.

Measurement Setup. To measure the impacts of tile size on culling performance, we implement a clipping-based frustum culling algorithm [58, 68] and an occlusion culling algorithm derived from ViVo [27] in Python. As for the compression efficiency, we employ Google Draco [4], a widely used volumetric video compression tool, with its default parameters. All experiments are conducted on an Apple M1 chip with 8 GB Memory.

Culling Accuracy. We first partition the *longdress* video into tiles with three different sizes and then perform a culling algorithm under 40 different viewport trajectories. To measure the culling accuracy, the volumetric video is culled at the smallest granularity, the point, which serves as the ground truth. The culling accuracy is quantified as the number of redundant points compared to the ground truth. The results show that the number of redundant points increases as tile size grows, as illustrated in Fig. 5.1. This occurs because the tile is regarded as visible if any of its points are not culled. Hence, when only a small part of a tile is watched, the entire tile is retained, leading

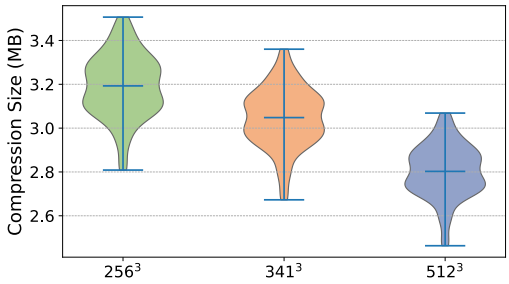


Figure 5.3: Compression size of different tile sizes.

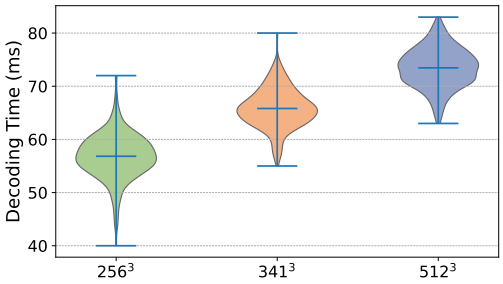


Figure 5.4: Decoding time of different tile sizes.

to a significant number of redundant points. Besides, the range of redundant points increases with the tile size. This suggests that larger tiles are more sensitive to user viewing patterns, potentially resulting in less efficient data transmission in dynamic watching behaviors.

Culling Time. The processing time of culling algorithms with various tile sizes is presented in Fig. 5.2. The results show that culling time decreases as tile size increases, primarily because larger tiles reduce the number of tiles that need to be processed. Additionally, smaller tiles exhibit higher variability in culling time compared to larger tiles. This happens because the smaller tiles introduce high uncertainty in the culling process by increasing the tile amounts. More tiles increase the sensitivity of the culling process to viewport changes and scene complexity, resulting in more dynamic and unpredictable culling times.

Compression efficiency. The compression efficiency is evaluated by analyzing the compressed data size for different tile sizes using Google Draco with default parameters, as Fig. 5.3 shows. The compressed data size decreases with the increase in tile size because larger tiles preserve more spatial relationships among points within the tile. These preserved relationships allow compression algorithms to exploit spatial redundancy more effectively, achieving higher compression ratios. In contrast, smaller tiles

segment the point cloud into small pieces, breaking many of these spatial relationships and hence reducing compression efficiency. Moreover, the distributions of compression sizes are similar for different tile sizes. It indicates that while the absolute compression efficiency varies with tile size, compression size is primarily determined by the frame contents rather than by the choice of tile size.

Compression time. The compression time of different tile sizes is illustrated in Fig. 5.4. The compression time increases as the tile size becomes large. Since the tile with a larger size contains more points and hence preserves more complex spatial relationships among points, it requires additional computational resources to decode the well-compressed data.

The above measurements demonstrate the complex impact of the tile size on culling and compression performance. These effects do not always align, making optimal tile size selection a challenging task. For example, though the small tile size reduces data redundancy and hence decreases the data to transmit, it incurs longer culling time, which potentially brings extra stalls during playback. On the other hand, small tiles decrease compression efficiency, resulting in larger compression sizes. However, they also require less decoding time, which can benefit real-time streaming. Such complicated trade-offs highlight the need for a dynamic and adaptive approach to tile size selection that considers the interplay of these factors during volumetric video streaming.

5.2.2 Optimal Tile Size Variability

In this subsection, we provide a toy example to further illustrate the advantages of flexible tile sizes in volumetric video streaming. In this toy example, we compare the volumetric video streaming with three fixed tile sizes, 256×256×256, 341×341×341 and 512×512×512, and a flexible tile size.

Table 5.1: Frame information for viewport 1 in the toy example

Tile Size	Transmission Size (MB)	Culling Time (ms)	Decoding Time (s)
256^3	0.3	15	10
341^3	0.7	6	19
512^3	1.3	2	30

Table 5.2: Frame information for viewport 2 in the toy example

Tile Size	Transmission Size (MB)	Culling Time (s)	Decoding Time (s)
256^3	3.0	18	52
341^3	2.8	8	60
512^3	2.6	3	78

To better demonstrate the benefit of flexible tile size under dynamic viewports, we select two distinct viewports. Viewport 1 (V1) focuses on a part of a specific region of the volumetric video and hence the culling performance varies significantly across different tile sizes, as shown in Table 5.1. Conversely, Viewport 2 (V2) covers the entire volumetric video, leading to similar culling performance under different tile sizes, as illustrated in Table 5.2. Moreover, we compare the streaming performance of different tile sizes under two bandwidth conditions, a low bandwidth (LB) throughput of 50 Mbps and a high bandwidth (HB) throughput of 200 Mbps.

We evaluate the streaming performance using the system latency, as the bitrate allocation is not taken into consideration in this toy example. The total latency of each strategy under four conditions is illustrated in Fig. 5.5. The tile size of 256^3 achieves the lowest latency under the viewport 1 regardless of bandwidth conditions. The high culling accuracy of the small tile size significantly reduces the amount of transmission data. On the contrary, the large tile size, 512^3, has a large amount of redundant data, leading to long transmission and decoding times. Hence, the fast culling process of a large tile size becomes trivial in these situations. The large tile size shows its advantages under the viewport 2 with a low bandwidth throughput. The tile size does not influence

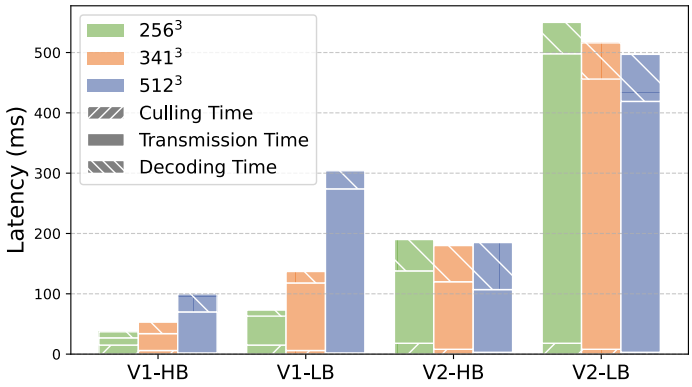


Figure 5.5: The system latency for different tile sizes.

the culling accuracy under the viewport 2 but the high compression efficiency of a large tile size reduces transmission data size. The reduction becomes especially important when the bandwidth condition is poor. The middle tile size, 341^3, has the lowest latency under the viewport 2 with a high bandwidth throughput. Though the middle tile size increases transmission latency compared to the small tile size, it decreases the culling and decoding latency.

To sum up, the optimal tile size is highly dynamic, varying with viewport patterns, bandwidth conditions, and their intricate interplay with culling performance and compression efficiency. These observations highlight the need to introduce flexible tile sizes into volumetric video streaming. It enables the streaming scheme to adapt to diverse conditions effectively.

Moreover, bitrate allocation is critical in video streaming. Transmitting volumetric video at the highest quality can result in excessive latency even with the optimal tile size. For example, under the V2-LB situation, the latency is up to 500 ms with the optimal tile size 512^3, which is impractical for real-time applications. This highlights the importance of a comprehensive tile configuration in volumetric video streaming, including tile size selection and tile bitrate allocation.

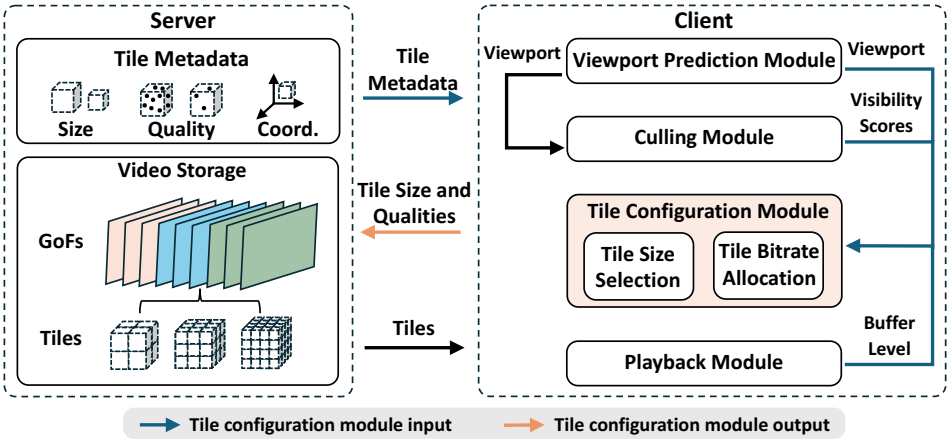


Figure 5.6: The overview of volumetric video streaming system with flexible tile sizes.

5.3 System Model and Problem Formulation

In this section, we present the system model, which is structured into four key components: the system overview, the tile-based volumetric video model, the volumetric video streaming model, and the video QoE model. Based on these models, we formulate the tile configuration as a QoE optimization problem.

5.3.1 System overview

An overview of the system is illustrated in Fig. 5.6. The system consists of a **content server** and a **client**. The content server stores and transmits tile-based volumetric videos, while the client decides on tile configuration to maximize QoE during playback.

On the server side, the volumetric video is pre-segmented into multiple groups of frames (GoF), with each GoF containing a fixed number of frames. Each frame is further partitioned into tiles of different sizes. These tiles are then sampled at multiple point cloud densities, representing varying quality levels, to support adaptive transmission. Additionally, the server maintains metadata for each tile, such as the tile size, quality level, and spatial coordinate. This information is later transmitted to the client for tile

configuration.

The client is equipped with several key modules to facilitate adaptive streaming. The core component is the **tile configuration module** which is responsible for deciding on *tile size selection* and *tile bitrate allocation*. To support these decisions, the module requires comprehensive input, including the predicted viewport, buffer levels, visibility scores, and tile metadata. While tile metadata is provided by the content server, other information is obtained from other local modules, ensuring user privacy during watching.

The **viewport prediction module** predicts the future viewport based on historical interaction data and motion patterns. The predicted viewport is necessary for the **culling module** to generate visibility scores. The visibility scores are calculated by identifying tiles that are within the FoV and detecting those that are occluded. The **playback module** monitors the current buffer level to ensure smooth streaming and seamless playback. Together with the tile information from the server, the client determines the optimal tile sizes and quality levels. Then it sends the tile request to the server and waits for video transmission.

5.3.2 Tile-based Volumetric Video Model

In this subsection, we describe the representation of volumetric video in detail. We first give the volumetric video model and viewport model, followed by the tile model and culling model.

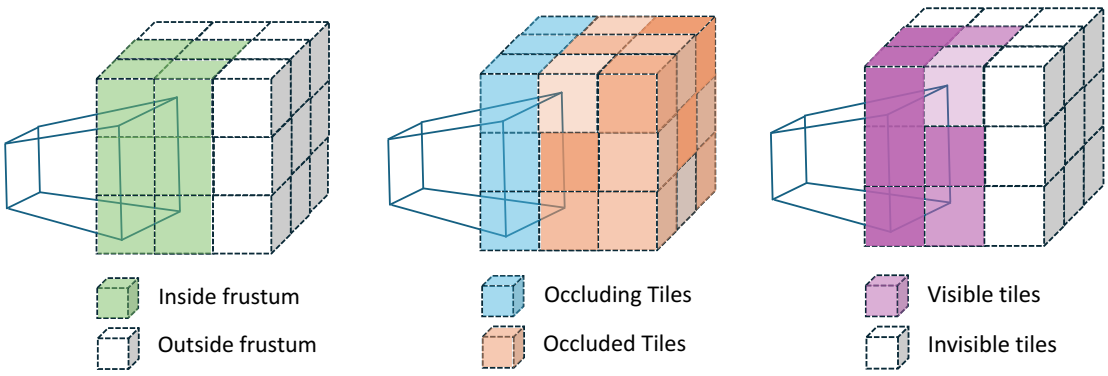
Video and viewport model. The volumetric video is conceptualized as a sequence of point cloud frames and is evenly divided into N^G groups of frames (GoFs) $\mathcal{G} = \{g_1, g_2, ..., g_{N^G}\}$. The data size of the GoF g_i is d_i. Each GoF is comprised of N^F frames $\mathcal{F}_i = \{f_{i1}, f_{i2}, ..., f_{iN^F}\}$. Each frame is bounded by a bounding box b_i^F, and hence the bounding box for frames in a GoF is $\mathcal{B}_i = \{b_{i1}^F, b_{i2}^F, ..., b_{iN^F}^F\}$. The total playback duration of

the GoF is τ^{PG}, and hence the duration of the volumetric video is the sum of all GoF durations $\tau^{PV} = N^G \tau^{PG}$.

Considering the interaction feature of the volumetric video, user viewports are an essential part of volumetric video streaming. The volumetric video provides 6 DoF for viewports, containing translation and rotation along three dimensions x, y, z-axis respectively. The viewport v_{ij} differs from frame to frame. The viewport prediction module predicts the user viewports for the next GoF $\hat{\mathbf{v}}_i = \{\hat{v}_{i1}, \hat{v}_{i2}, ..., \hat{v}_{iN^F}\}$ based on viewports history in the last K GoFs $\mathbf{v}_{iK}^H = \bigcup_{j=i-K}^{i-1} \mathbf{v}_j$.

Tile model. The tile is the minimal unit to store and transmit the volumetric video. A tile within the frame f_{ij} in GoF g_i is presented as a cuboid box b_{ijk} with eight points. There are N^{TS} tile sizes to select at the server, $\mathcal{Z} = \{z_1, z_2, ..., z_{N^{TS}}\}$. We assume the tile size remains the same in a GoF for simplification. With each tile size z, the volumetric video frame is evenly partitioned into N_z^{TF} tiles. Hence a GoF with tile size z contains $N_z^{TG} = N_z^{TF} \cdot N^F$ tiles. For each tile, we can set a different point cloud density (bitrate) q for it from the density set $\mathcal{Q} = \{q_1, q_2, ..., q_{N^Q}\}$. The minimum density is $q_1 = 0$, which means the tile is not transmitted, while the maximum density is $q_{N^Q} = 1$. The data size and decoding time of the tile b_{ijk} with density q_l are respectively denoted by $d_{ijk}(q_l)$ and $\tau_{ijk}^{DC}(q_l)$. Specifically, we define the original data size of the tile b_{ijk} as d_{ijk}^{TO}, which is equal to the data size of the maximum density $d_{ijk}^{TO} = d_{ijk}(q_N)$. Hence, the data size set of all tiles with the size z in a GoF g_i is $\mathcal{D}_i(z) = \{d_{ijk}^{TO} | 1 \le j \le N^F, 1 \le k \le N_z^{TF}\}$.

Culling model. The culling module calculates the visibility score $x_{ijk}^V(\hat{v}_{ij})$ for each tile b_{ijk} based on the predicted viewport $\hat{v}_{ij}$ of the frame f_j in the GoF g_i. It employs two methods: frustum culling and occlusion culling, as illustrated in Fig. 5.7. Frustum culling eliminates tiles outside the viewing frustum, whereas occlusion culling identifies tiles that are occluded by other tiles.



(a) Frustum culling. (b) Occlusion culling. (c) Visibility score.

Figure 5.7: Culling methods and visibility scores illustration.

A clipping-based frustum culling technique [21] is employed. Tiles are projected into the clip space, and the frustum visibility is calculated by determining whether it is completely outside the view window. Fig. 5.7(a) shows the frustum culling results. A binary variable is used to indicate the frustum visibility score $x^F_{ijk}(\hat{v}_{ij})$,

$$x^F_{ijk}(\hat{v}_{ij}) = \begin{cases} 1, & \text{if tile is inside,} \\ 0, & \text{if tile is outside.} \end{cases} \tag{5.1}$$

We leverage the occlusion culling method in Vivo [27] because the traditional occlusion culling method is not suitable for tile-based volumetric videos. The problem arises from the traditional occlusion culling method because points from a rear tile may still remain visible, not fully obscured by points from the tile in front. Fig. 5.7(b) represents the occlusion culling results. Though tiles in the right two columns are occluded by tiles in the first column, the degree of occlusion varies, as indicated by the color of tiles. Therefore, we use occlusion levels, $x^O_{ijk}(\hat{v}_{ij})$, instead of a binary variable to represent occlusion visibility. The occlusion level $x^O_{ijk}(\hat{v}_{ij})$ is calculated according to the number of occluding tiles $N^O_{ijk}(\hat{v}_{ij})$ and the ratio between the highest point count among those

occluding tiles to the point count of the given tile $r_{ijk}(\hat{v}_{ij})$,

$$x_{ijk}^{O}(\hat{v}_{ij}) = \begin{cases} 1, & \text{if } r_{ijk}(\hat{v}_{ij}) < th(\alpha_0, \beta) \\ 0.8, & \text{if } th(\alpha_0, \beta) \le r_{ijk}(\hat{v}_{ij}) < th(\alpha_1, \beta) \\ 0.6, & \text{if } th(\alpha_1, \beta) \le r_{ijk}(\hat{v}_{ij}) < th(\alpha_2, \beta) \\ 0.4, & \text{if } th(\alpha_2, \beta) \le r_{ijk}(\hat{v}_{ij}) \le th(\alpha_3, \beta) \\ 0.2, & \text{if } th(\alpha_3, \beta) \le r_{ijk}(\hat{v}_{ij}) \end{cases} \tag{5.2}$$

$$th(\alpha, \beta) = \alpha \beta^{N_{ij}^{O}(\hat{v}_{ij})-1}, \tag{5.3}$$

where $\alpha_0, \alpha_1, \alpha_2, \alpha_3, \beta$ are parameters to adjust the value of the occlusion score, and $th(\alpha, \beta)$ is used to measure the occlusion degree. Given the inherent difficulty in achieving complete occlusion between tiles, we set the minimal occlusion visibility score as 0.2 rather than 0.

After culling, the visibility score of each tile is the product of two scores $x_{ijk}^{V}(\hat{v}_{ij}) = x_{ijk}^{F}(\hat{v}_{ij}) \cdot x_{ijk}^{O}(\hat{v}_{ij})$. Tiles with $x_{ijk}^{V}(\hat{v}_{ij}) = 0$ can be removed due to their invisibility, as suggested in Fig. 5.7(c). Hence, the visibility score of the GoF is $\mathcal{X}_i^V = \{x_{ijk}^{V} | 1 \le j \le N^F, 1 \le k \le N_z^{TF}\}$. The total culling time for the GoF is τ_i^C.

5.3.3 Volumetric Video Streaming Model

In this subsection, we outline the streaming process and buffer dynamics to provide a better understanding of volumetric video delivery and playback status.

Video streaming model. The streaming process is divided into several slots, denoted by $\mathcal{T} = \{t_1, t_2, ..., t_i, ...\}$, as illustrated in Fig. 5.8. At the beginning of each slot t_i, the server first sends the tile information with varying sizes to the client (at t_1 in the figure), including coordinates of tile cuboid boxes and data amount of each tile with different qualities. The client predicts the viewport of the new GoF $\hat{\mathbf{v}}_i$, selects an appropriate tile

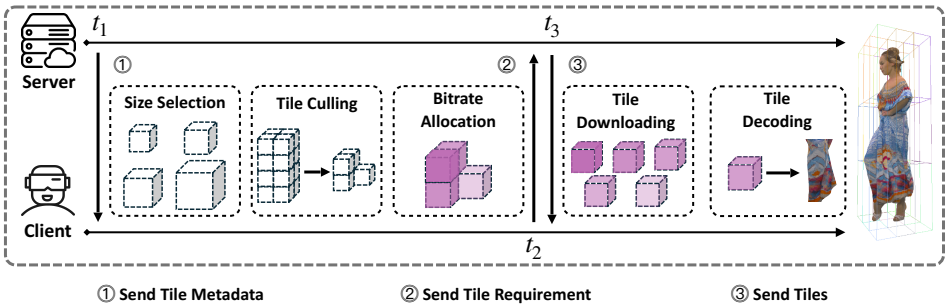


Figure 5.8: Volumetric video with flexible tile sizes streaming process.

size z_i, and culls the volumetric video based on them. Given the culling results, the client allocates point cloud densities to each tile $\mathbf{q}_i = \{q_{ijk} | 1 \le i \le N^G, 1 \le j \le N^F, 1 \le k \le N_z^{TF}\}$ and sends these requests back to the server (at t_2 in the figure). Then, the server transmits tiles as requested (at t_3 in the figure). Therefore, the total download duration of the GoF $\tau_i^{DL}(\mathbf{q}_i)$ is calculated by dividing all requested tile sizes by the average bandwidth c_i in the current slot,

$$\tau_i^{DL}(\mathbf{q}_i) = \frac{\sum_{j=1}^{N^F} \sum_{k=1}^{N_z^{TF}} d_{ijk}(q_{ijk})}{c_i}. \tag{5.4}$$

Before display, the received frames need to be decoded at first. The decoding duration $\tau_i^{DC}(\mathbf{q}_i)$ is the sum of the decoding time of each tile $\tau_{ij}^{DC}(q_{ij})$,

$$\tau_i^{DC}(\mathbf{q}_i) = \sum_{j=1}^{N^F} \sum_{k=1}^{N_z^{TF}} \tau_{ijk}^{DC}(q_{ijk}). \tag{5.5}$$

After the completion of the download and decoding process, the GoF is saved to the buffer for display. The current slot ends once the GoF is stored in the buffer. A new slot starts immediately for the next GoF. Therefore, the slot duration τ_i is the sum of culling duration $\tau_i^C(z_i)$, downloading duration $\tau_i^{DL}(\mathbf{q}_i)$, and decoding duration $\tau_i^{DC}(\mathbf{q}_i)$,

$$\tau_i = \tau_i^C(z_i) + \tau_i^{DL}(\mathbf{q}_i) + \tau_i^{DC}(\mathbf{q}_i). \tag{5.6}$$

The tile configuration duration is relatively insignificant and hence can be ignored (detailed in Section 5.5).

Video buffer model. The client maintains a video playback buffer with the capacity of B_{max} to hold frames pending display. The buffer dynamics are computed as follows.

The buffer state at any time $B(t)$ is obtained by subtracting the count of played GoFs $n^P(t)$ from the number of total decoded GoFs $n^B(t)$,

$$B(t) = \min\{n^B(t) - n^P(t), B_{max}\}. \tag{5.7}$$

Given the playtime t_i^P and decoding completion time (the time when saved to the buffer) t_i^B of each GoF, the number of played GoFs $n^P(t)$ and decoded GoFs $n^B(t)$ can be calculated as,

$$\begin{aligned} n^P(t) &= \max\{i | t_i^P \leq t\} \\ n^B(t) &= \max\{i | t_i^B \leq t\}. \end{aligned} \tag{5.8}$$

The GoF g_i begins to play when the previous GoF g_{i-1} finishes playing except for the empty buffer. When the buffer is empty, the playback time of the GoF g_i is the time when it is saved to the buffer t_i^B, which is equal to its decoding completion time. Hence, the playback time t_i^P of the GoF g_i is determined by the maximum value of these two time points,

$$t_i^P = \max\{t_{i-1}^P + \tau^{PG}, t_i^B\}. \tag{5.9}$$

The decoding completion time t_i^B is the sum of the request time t_i^R, download duration τ_i^{DL}, and decoding duration τ_i^{DC},

$$t_i^B = t_i^R + \tau_i^{DL} + \tau_i^{DC}. \tag{5.10}$$

As mentioned before, a new GoF request is sent to the server after the tile configuration and culling process. Hence, the request time t_i^R is the sum of the slot start time t_i^S and the culling duration τ_i^C, as the tile configuration duration can be neglected,

$$t_i^R = t_i^S + \tau_i^C. \tag{5.11}$$

The new slot begins when the last slot ends, except when the buffer is full. If so, the slot start time will be postponed until there is room in the buffer. Hence, the slot start time t_i^S is the maximum value of the last slot completion time and the time when the buffer is no longer full,

$$t_i^S = \max\{t_{i-1}^S + \tau_{i-1}, t_{i-B_{max}}^P\}, \tag{5.12}$$

where $t_{i-B_{max}}^P$ is the playback time of the first GoF in the buffer at this slot.

5.3.4 Volumetric Video QoE Model

In this subsection, we introduce the QoE model for volumetric video. The QoE model is composed of video quality, quality switch impairment, and stall impairment. We first discuss these three components individually and then present the final QoE model.

Video quality model. Though the video quality is primarily measured by the average tile point cloud density, the occlusion effects should be taken into consideration as well. Intuitively, for two tiles with the same density, the tile with higher occlusion visibility scores has better visual quality than the tile with lower scores. Hence, we leverage the video quality model in the previous work [105]. It measures the tile quality by the ratio between the number of corresponding pixels rendered on the display screen at a given viewport and the number of 3D points, while we use the ratio between the point cloud density and visibility score $\frac{q_{ijk}}{x_{ijk}^V(v_{ij})}$ for approximation. Besides, the punishment for missing tiles is taken into consideration as well, denoted by the negative value of visibility score $-x_{ijk}^V(v_{ij})$. The tile quality $Q_{ijk}^T(q_{ijk}, v_{ij})$ of the tile b_{ijk} with the density

q_{ijk} is calculated as,

$$Q_{ijk}^{T}(q_{ijk}, v_{ij}) = \begin{cases} \min\{1, \frac{q_{ijk}}{x_{ijk}^{V}(v_{ij})}\}, & \text{if } x_{ijk}^{V}(v_{ij}) > 0, \\ & q_{ijk} \neq 0 \\ -x_{ijk}^{V}(v_{ij}), & \text{if } x_{ijk}^{V}(v_{ij}) > 0, \\ & q_{ijk} = 0 \\ 0, & \text{otherwise.} \end{cases} \tag{5.13}$$

The frame quality $Q_{ij}^{F}(\mathbf{q_{ij}}, v_{ij})$ is the average quality of all visible tiles within the frame,

$$Q_{ij}^{F}(\mathbf{q_{ij}}, v_{ij}) = \frac{\sum_{k=1}^{N_z^{TF}} Q_{ijk}^{T}(q_{ijk}, v_{ij})}{\sum_{k=1}^{N_z^{TF}} \mathbf{1}_{x_{ijk}^{V}(v_{ij})\neq 0}}, \tag{5.14}$$

where z is the tile size of this GoF and $\mathbf{1}_{x\neq 0}$ is the indicator function, which evaluates to 1 when $x \neq 0$, and 0 otherwise. The GoF quality $Q_{i}^{G}(\mathbf{q_i}, \mathbf{v}_i)$ is the average quality of all frames within it,

$$Q_{i}^{G}(\mathbf{q_i}, \mathbf{v}_i) = \frac{\sum_{j=1}^{N^F} Q_{ij}^{F}(q_{ij}, v_{ij})}{N^F}. \tag{5.15}$$

The video quality Q^V is the average quality of all GoFs,

$$Q^{V} = \frac{\sum_{i=1}^{N^G} Q_{i}^{G}(\mathbf{q_i}, \mathbf{v}_i)}{N^G}. \tag{5.16}$$

Quality switch impairment model. There are two types of video quality switch impairment: the intra-frame switch and the inter-frame switch. The intra-frame quality switch is caused by the different tile qualities within the frame, while the inter-frame quality switch is caused by the different qualities of successive frames. Similar to YuZu [120], we measure the intra-frame quality impairment I_{ij}^{intra} by the standard deviation of visible tile qualities, while the inter-frame quality impairment I_{ij}^{inter} is measured by

the absolute difference between successive frames.

$$I_{ij}^{intra} = \textbf{StdDev}(\{Q_{ijk}^{T} | x_{ijk}^{V} > 0\}) \tag{5.17}$$

$$I_{ij}^{inter} = |Q_{ij}^{F} - Q_{ij-1}^{F}| \tag{5.18}$$

where Q_{ijk}^{T} is short for $Q_{ijk}^{T}(q_{ijk}, v_{ij})$, x_{ijk}^{V} is short for $x_{ijk}^{V}(v_{ij})$, and Q_{ij}^{F} is short for $Q_{ij}^{F}(\mathbf{q}_{ij}, v_{ij})$. In each GoF, the quality switch impairment I_i^G is the average of all frames,

$$I_i^G = \frac{\sum_{j=1}^{N^F} I_{ij}^{intra} + I_{ij}^{inter}}{N^F}. \tag{5.19}$$

Hence, the quality switch impairment I^Q of the entire video is the average of all GoFs,

$$I^Q = \frac{\sum_{i=1}^{N^G} I_i^G}{N^G}. \tag{5.20}$$

Stall impairment. A video stall happens when the video player fails to retrieve the next GoF from the buffer in time. We use the total stall time to measure the stall impairment. Since the new GoF is expected to play after the last GoF, the expected playback time t_i^{EP} is the sum of the last GoF playback time t_{i-1}^{EP} and the GoF duration τ^{PG},

$$t_i^{EP} = t_{i-1}^{P} + \tau^{PG}. \tag{5.21}$$

The stall duration τ_i^S of the frame f_i can be straightforwardly achieved by the difference between the playback time t_i^P and the expected playback time t_i^{EP},

$$\tau_i^S = t_i^P - t_i^{EP}. \tag{5.22}$$

Thus, the cumulative stall duration τ^S is the average of all frame stalls,

$$\tau^S = \frac{\sum_{i=1}^{N^G} \tau_i^S}{N^G}. \tag{5.23}$$

For better evaluation, we employ the ratio between the average stall duration τ^S and the GoF duration τ^{PG} as the stall impairment I^S,

$$I^S = \frac{\tau^S}{\tau^{PG}}. \tag{5.24}$$

Video QoE model. The final QoE metric QoE is defined as the weighted sum of video quality Q^V, quality switch impairment I^Q, and stall impairment I^S,

$$QoE = Q^V - \omega_1 I^Q - \omega_2 I^S, \tag{5.25}$$

where ω_1 and ω_2 are two non-negative weighting parameters. These parameters allow the model to be tailored to specific applications or user preferences.

5.3.5 Problem Formulation

Based on the above system models, we formulate the Volumetric Video Tile Configuration (VV-TC) problem to maximize the long-term QoE by deciding on tile size selection and tile bitrate allocation,

$$\max_{\mathbf{s},\mathbf{q}} QoE = Q^V - \omega_1 I^Q - \omega_2 I^S \tag{5.26}$$

$$\text{s.t.} \quad s_i \in \mathcal{S}, \forall s_i \in \mathbf{s} \tag{5.27a}$$

$$q_{ijk} \in \mathcal{Q}, \forall q_{ijk} \in \mathbf{q}. \tag{5.27b}$$

The main challenge in solving the VV-TC problem lies in the uncertainty of future bandwidth and viewport. Even with available future information, the problem remains complex due to the interplay between video quality, network conditions, and user interactions, which impact buffer dynamics and QoE. To address these challenges, we first re-model the VV-TC problem as a Markov Decision Process (MDP) in Section 5.4.2. This MDP formulation makes it possible to solve the complex VV-TC problem time slot by time slot. Then we propose an HRL-based approach to jointly determine the tile size selection and bitrate allocation in Section 5.4.3.

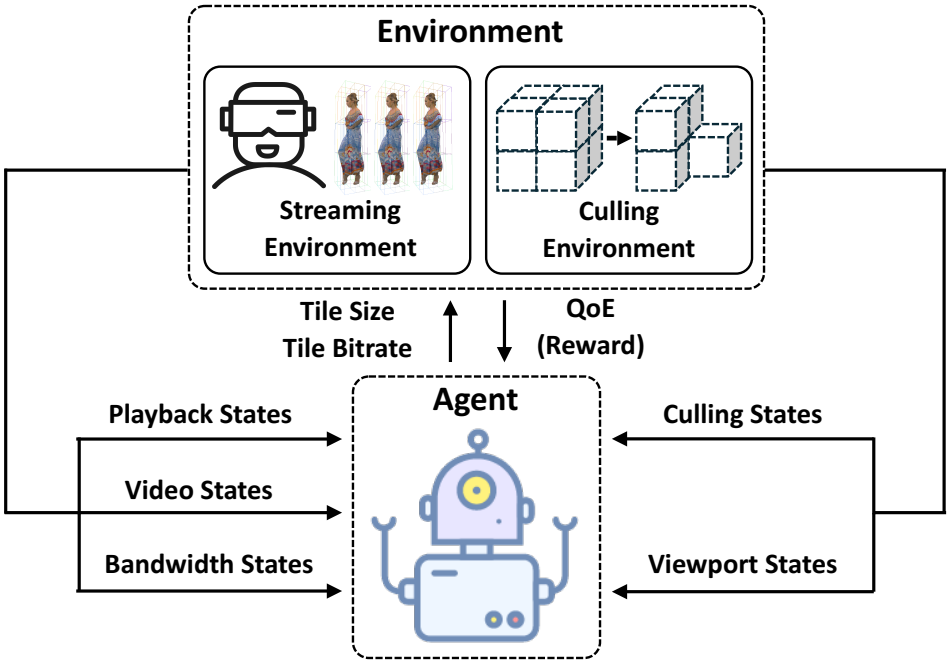


Figure 5.9: The overview of agent-environment interaction.

5.4 Hierarchical Reinforcement Learning Framework for VV-TC Problem

In this section, we propose a hierarchical reinforcement learning framework to address the VV-TC problem. The HRL framework jointly optimizes tile size selection and tile bitrate allocation to maximize the long-term QoE.

5.4.1 HRL Framework Overview

We first decouple the VV-TC problem into a series of single-slot QoE optimization problems, which allow the HRL framework to make decisions time slot by time slot. As illustrated in Fig. 5.9, at each time slot, the HRL framework collects information from both the culling environment and the streaming environment. The culling environment provides *culling states* and *viewport states*, while the streaming environment offers *playback states*, *video states*, and *bandwidth states*. Then it selects the tile size and

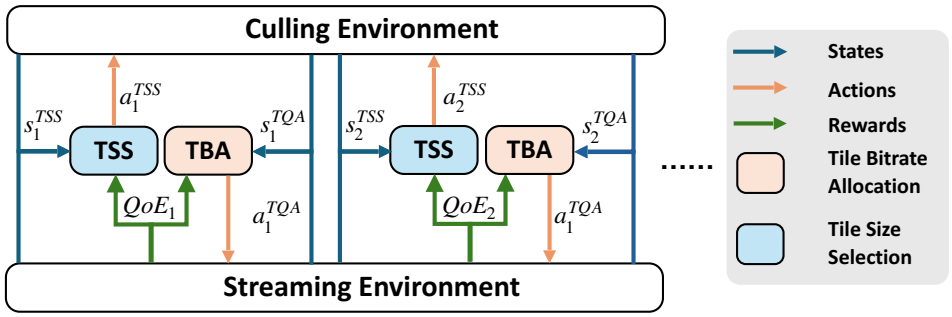


Figure 5.10: Hierarchical reinforcement learning framework overview.

allocates bitrate to each tile and receives the QoE feedback from the environment.

It is challenging to solve the single-slot QoE optimization problem with a single RL agent due to the inherent dependency between tile size selection and bitrate allocation. The tile size directly influences culling performance such as visibility scores and culling time, which subsequently impacts tile bitrate allocation. To address this, we propose an HRL framework, decoupling the problem into two sequential stages, the *tile size selection stage* and the *tile bitrate allocation stage*, as shown in Fig. 5.10. In the first stage, the TSS agent collects environment states and selects the appropriate tile size for culling. In the second stage, the TBA agent allocates point cloud density to each tile according to the information obtained from the culling and streaming environment. Both agents receive the same rewards, i.e., QoEs, after tile bitrate allocation.

5.4.2 Markov Decision Process Formulation for HRL Framework

To train the TSS and TBA agents in the HRL framework, we first need an MDP for each stage. An MDP is defined by a state space $\mathcal{S}$, an action space $\mathcal{A}$, a transition probability function $\mathcal{P}(s_{i+1}|s_i,a_i)$ and a reward function $\mathcal{R}(s_i,a_i)$. The objective of the agent is to learn a policy $\pi(a_i|s_i)$, a mapping from states to actions, that maximizes the expected sum of discounted rewards over time.

5.4.2.1 The Tile Size Selection Stage

In this stage, the TSS agent aims to select an appropriate tile size for culling. We formulate the tile size selection problem in the form of an MDP as follows.

State. The state space consists of a set of observations of the environment. The culling environment provides *culling states* and *viewport states*, while the streaming environment offers *playback states*, *video states* and *bandwidth states*.

Specifically, the *culling states* and *viewport states* help the agent to predict current culling performance by providing recent culling process performance and viewports history and prediction. It includes the culling time incurred by the culling module for the past K GoFs $\mathbf{T}^C_{iK} = \{\tau^C_{i-K}, ..., \tau^C_{i-1}\}$, the tile size $\mathbf{Z}_{iK} = \{z_{i-K}, ..., z_{i-1}\}$, the ratio between the number of points before and after culling $\mathbf{R}_{iK} = \{r_{i-K}, \cdots, d_{i-1}\}$, and corresponding visibility scores $\mathbf{X}^V_{iK} = \bigcup_{j=i-K}^{i-1} \mathcal{X}^V_j$. To further support the learning of culling performance, *viewport states* provide the viewports history $\mathbf{V}^H_{iK}$ and the predicted viewport $\hat{\mathbf{v}}_i$.

The *playback states* tells the agent about the current buffer level $B(t_i)$, while the *video states* provides the bounding box of each frame in the past K GoFs $\mathbf{B}_{iK} = \bigcup_{j=i-K}^{i-1} \mathcal{B}_i$, and the data size of tiles with different sizes in next GoF $\mathbf{D}_i = \bigcup_{z \in \mathcal{Z}} \mathcal{D}_i(z)$. The *bandwidth states* describe recent bandwidth conditions, including the average bandwidth throughput for the past K slots $\mathbf{C}_{iK} = \{c_{i-K}, \cdots, c_{i-1}\}$ and the downloading duration $\vec{\tau}^{DL}_{iK} = \{\tau^{DL}_{i-K}\}$.

To sum up, the states in the tile size selection stage s^{TSS}_i at the slot t_i is defined as a tuple of all these states,

$$s^{TSS}_i = \{\mathbf{T}^C_{iK}, \mathbf{Z}_{iK}, \mathbf{R}_{iK}, \mathbf{X}^V_{iK}, \mathbf{V}^H_{ik}, \hat{\mathbf{v}}_i, B(t_i), \mathbf{B}_{iK}, \mathbf{D}_i, \mathbf{C}_{iK}, \vec{\tau}^{DL}_{iK}\}. \tag{5.28}$$

Action and Reward. The action space is the tile size set $\mathcal{Z}$, and hence the action in this stage a^{TSS}_i is to select one tile size z from the tile size set $\mathcal{Z}$.

The reward is defined as the QoE of each GoF Q_i^G,

$$r_i^{TSS} = QoE_i = Q_i^G - \omega_1 I_i^G - \omega_2 \frac{\tau_i^S}{\tau^{PG}}. \tag{5.29}$$

However, the reward of the tile size selection stage can only be obtained once the tile bitrate is determined. Hence, to train the TSS agent in the tile size selection stage, a pre-trained TBA agent in the tile bitrate allocation stage is required, details in the Section 5.4.3.

5.4.2.2 The Tile Bitrate Allocation Stage

The tile bitrate allocation stage is after the tile size selection stage. Given the tile size for the new GoF, point cloud densities are allocated to each tile according to their visibility scores, bandwidth conditions, and playback states. The objective of this stage is to ensure the quality of the visual experience while minimizing unnecessary data transmission and quality switch impairment, thus efficiently balancing video quality and bandwidth utilization.

State. The *culling states* comes from the culling environment, while the *playback states*, *video states* and *bandwidth states* are given by the streaming environment. The *culling states* provide the selected tile size z, visibility scores for the new GoF $\mathcal{X}_i^V$ calculated by the culling module, and the culling duration for the new GoF τ_i^C. The *playback states* include the current buffer level $B(t_i)$, the number of the remaining undownloaded GoFs $n_i^{UD} = N^G - i$, and the average QoE of the previous GoF Q_{i-1}^G. The *video states* are made up of the data size of each tile in the next Gof $\mathcal{D}_i(z)$. The *bandwidth states* are the same as the one in the tile size selection stage, including the average throughput $\mathbf{C}_{iK}$ and downloading durations $\vec{\tau}_{iK}^{DL}$ in the past K slots. Overall, the state in the tile bitrate allocation stage s_i^{TBA} at the slot t_i is defined as the tuple of all above states,

$$s_i^{TBA} = \{z, \mathcal{X}_i^V, \tau_i^C, B(t_i), n_i^{UD}, Q_{i-1}^G, \mathcal{D}_i(z), \mathbf{C}_{iK}, \vec{\tau}_{iK}^{DL}\}. \tag{5.30}$$

Action and Reward. Given the culling results, the action in the tile bitrate allocation stage is to allocate point cloud density for each tile. The density is selected from the density set $\mathcal{Q}$. Hence, the action a_i^{TBA} equals the tile bitrate $\mathbf{q}_i$ in the following GoF g_i.

After the action is executed, the environment gives feedback in the form of a reward. The reward r_i^{TBA} is defined as the QoE of the next GoF as Eq. (5.29) shows. Since the reward functions for both the tile size selection and tile bitrate allocation stages are identical, we denote the reward by r_i.

5.4.3 Training Process of HRL Framework

Given the formulated MDPs, which define the training inputs, outputs, and rewards, we provide the detailed training process for the HRL framework in this subsection. To ensure efficient learning and stable convergence, we employ the Proximal Policy Optimization algorithm [87] in both the TSS agent and TBA agent. Additionally, a two-stage reverse training methodology is applied in the HRL framework to solve the VV-TC problem.

5.4.3.1 PPO Algorithm

The PPO algorithm is a state-of-the-art reinforcement learning method widely applied in various decision-making problems due to its balance of stability and efficiency. As shown in Fig. 5.11, PPO contains two neural networks: the *actor network* with the parameters θ, and the *critic network* with the parameters μ. The updates of these two networks are interdependent. The *actor network* relies on the value estimates provided by the *critic network* to compute the advantage function, while the *critic network* updates its parameters by the differences between the estimation and the actual value under the current policy of the *actor network*.

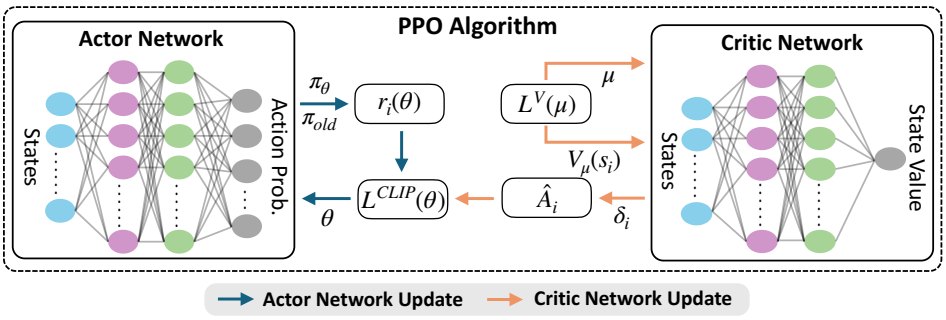


Figure 5.11: The PPO algorithm.

NN architecture. The *actor network* is used to select the action from the action space according to the policy π_θ. Specifically, the TSS agent selects the tile size, while the TBA agent allocates point cloud density to each tile. The *critic network* evaluates the expected return of the current state following the policy. They have the same input layer, representing current state information. Meanwhile, the hidden layers of them are implemented via fully-connected (FC) layer structures, followed by the tanh activation function. The *actor network* outputs the probability distribution over all possible actions through a softmax activation function. On the other hand, the *critic network* outputs the value estimation of the current state following the current policy.

Training methodology. In PPO, the *actor network* updates the policy by maximizing a clipped surrogate objective function, which limits the extent of policy updates to stay within a predetermined range.

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_i \left[\min(r_i(\theta)\hat{A}_i, \text{clip}_\epsilon(r_i(\theta))\hat{A}_i) \right], \tag{5.31}$$

where $r_i(\theta) = \frac{\pi_\theta(a_i|s_i)}{\pi_{\theta_{old}}(a_i|s_i)}$ is the action probability ratio taken by the new and old policy, $\hat{A}_i$ is an estimated advantage function, and $\text{clip}_\epsilon(r_i(\theta))$ is a clipping function to restrict $r_i(\theta)$ between $1-\epsilon$ and $1+\epsilon$. The advantage function measures the benefit brought by the action a_i in the state s_i by calculating the difference between the expected return after taking the action a_i in the state s_i and the expected return with the current policy. Generalized Advantage Estimation (GAE) [86] is used in our model. GAE calculates the

advantage estimates using a combination of Temporal Difference (TD) residuals across multiple steps,

$$\delta_i = r_t + \gamma V_\mu(s_{i+1}) - V_\mu(s_i). \tag{5.32}$$

The network parameters are updated iteratively with the loss function gradient,

$$\theta = \theta_{old} + \alpha \nabla_\theta L^{CLIP}(\theta), \tag{5.33}$$

where α is the learning rate.

The *critic network* is updated to minimize the difference between its predictions and the actual returns observed from the environment. Hence, the loss function is the Mean Squared Error (MSE) of current value estimation $V_\mu(s_i^{TBA})$ and the discounted sum of future rewards V_i^T,

$$L^V(\mu) = \text{MSE}(V_\mu(s_i^{TBA}), V_i^T). \tag{5.34}$$

The *critic network* parameters are updated with the loss function gradient as well,

$$\mu = \mu_{old} - \alpha \nabla_\mu L^V(\mu), \tag{5.35}$$

The training process of the PPO algorithm is shown in Alg. 5.1. At first, both the network parameters and environment are initialized (lines 1-2). The old policy is recorded for further update (line 3). The updating process is iteratively executed across numerous iterations. In each iteration, a set of trajectories is collected, and both the actor and critic network update their parameters for every minibatch extracted from the collected trajectories (lines 4-12).

5.4.3.2 Two-stage reverse training methodology

It is infeasible to train the TSS agent and TBA agent with the PPO algorithm directly. On the one hand, the TSS agent can not obtain the reward until the tile bitrate is allocated. On the other hand, the TBA agent can not make decisions without the specific tile size. To address this, we adopt a reverse training methodology for the two RL agents.

Algorithm 5.1 Training progress of PPO Algorithm

Input: Environment

Initialize the actor and critic network with random parameters θ and μ
Initialize environment and get states s_0, $s_i \leftarrow s_0$
Record the old policy parameter $\theta_{old} \leftarrow \theta$
foreach *iteration* **do**
    Run policy $\pi(\theta)$ for N^T timesteps in environment and collect trajectories $\{s_i, a_i, r_i\}$
    Estimate advantage function $\hat{A}_t$ with critic network by Eq. (5.32)
    foreach *minibatch in trajectories* **do**
        Update policy parameters θ with Eq. (5.33)
        Update critic parameters μ with Eq. (5.35)
    end
    $\theta_{old} \leftarrow \theta$
end

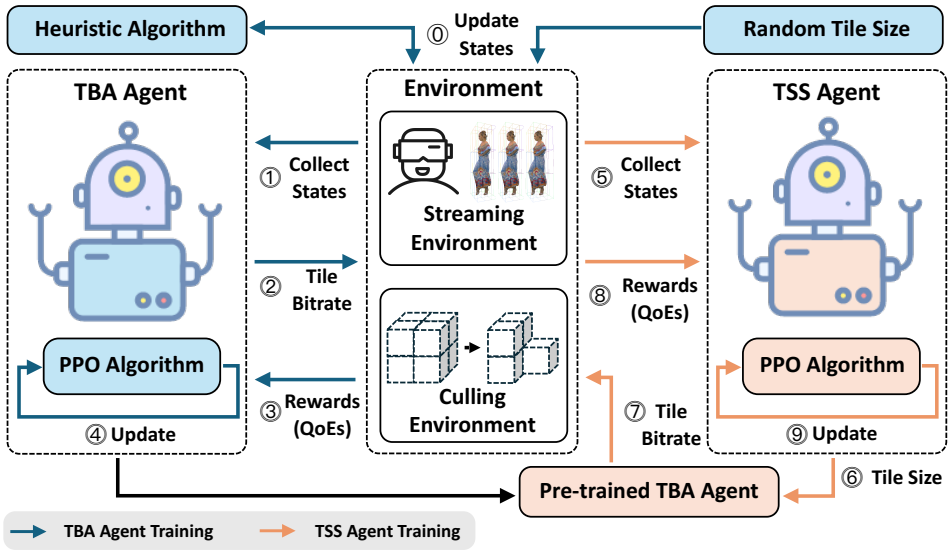


Figure 5.12: The training process in the HRL approach.

The two-stage reverse training process is illustrated in Fig. 5.12. Initially, the TBA agent is trained independently using random tile sizes to optimize the bitrate allocation without relying on prior knowledge of the optimal tile size selection. Then, the pre-trained TBA agent is integrated into the training of the TSS agent. The joint decisions of the TSS agent and TBA agent generate a reward for the TSS agent to learn optimal tile size selection policies.

TBA agent training. While the random tile sizes eliminate the dependency of tile size selection, the large action space creates a bottleneck for the TBA agent to explore solutions efficiently, especially when smaller tile sizes are selected. This challenge becomes more significant in volumetric video streaming because it is a more complicated environment. Hence, it can result in poor convergence and limited performance if we directly apply RL to such a high-dimensional problem. To overcome this limitation, we incorporate a heuristic algorithm to guide the training process of RL. The heuristic algorithm provides an initial suggestion for tile bitrate allocation, effectively facilitating convergence and improving RL performance.

Specifically, a random tile size is selected for the culling module to generate visibility scores. Then the visibility scores and other environment states are fed to a heuristic algorithm to produce an initial tile bitrate allocation $\mathbf{q}_i^{HEU}$. The initial solution $\mathbf{q}_i^{HEU}$ together with other environment observations s_i^{TBA} form the new states for the TBA agent,

$$s_i^{TBA'} = s_i^{TBA} \cup \{\mathbf{q}_i^{HEU}\}. \tag{5.36}$$

Then, the TBA agent follows the training process of the PPO algorithm to update its actor and critic network parameters. It collects the states $s_i^{TBA'}$, allocates tile bitrate with the current policy π_θ^{TBA}, and receives QoEs. The policy and critic parameters are updated according to the Eq. (5.33) and Eq. (5.35) until it converges.

TSS agent training. The pre-trained TBA agent is integrated into the training process. Initially, the environment states are observed by the TSS agent. Then it selects a tile size based on the current state s_i^{TSS} and policy π_θ^{TSS}. Following this, the culling module calculates visibility scores for each tile, which are updated in the states for the pre-trained TBA as well. Given the tile bitrate allocation, the environment gives QoEs back to the TSS agent for network training.

Algorithm 5.2 HRL-based Tile Configuration Framework

Input: Trained TBA agent and TSS agent

1 Initialize environment s_0^{TSS}, s_0^{TBA}
2 $s_i^{TSS} \leftarrow s_0^{TSS}$, $s_i^{TBA} \leftarrow s_0^{TBA}$
3 **while** *Video is not fully downloaded* **do**
4 $\quad a_i^{TSS} \leftarrow \pi_\theta^{TSS}(s_i^{TSS})$
5 $\quad$ Execute culling methods based on the selected tile size a_i^{TSS}
6 $\quad$ Update the TBA state s_i^{TBA}: tile size $z = a_i^{TSS}$, culling time $\tau_i^C(z)$, visibility score $\mathcal{X}_i^V$
7 $\quad a_i^{TBA} \leftarrow \pi_\theta^{TBA}(s_i^{TBA})$
8 $\quad$ Require a new GoF with the tile size a_i^{TSS} and tile qualities a_i^{TBA}
9 $\quad$ Observe reward and new states $r_i, s_{i+1}^{TSS}, s_{i+1}^{TBA}$
10 $\quad s_i^{TSS} \leftarrow s_{i+1}^{TSS}$, $s_i^{TBA} \leftarrow s_{i+1}^{TBA}$
11 **end**

HRL Interaction with the Environment. Once the HRL framework is fully trained, it interacts dynamically with the environment to optimize tile configuration decisions, as Alg. 5.2 shows. At the beginning of each video, the environment states are initialized and observed (lines 1-2). At each time slot, the TSS agent first observes environment states and selects a tile size for the culling module (lines 4-5). Then, the TBA agent obtains relative observations and allocates bitrate to tiles (lines 6-7). The client then requires a new GoF with the tile size and bitrate, receives the QoE for this GoF, and collects new states (lines 8-10). The process repeats until the whole video is downloaded.

5.5 Evaluation

In this section, we evaluate the performance of the HRL-based approach through trace-driven simulations, to answer the following questions:

- How does the heuristic algorithm help RL to facilitate training and improve performance (Section 5.5.2)?

- What is the performance of the HRL-based approach with flexible tile size compared to other volumetric video streaming frameworks (Section 5.5.3)?
- What is the performance of the HRL-based approach with flexible tile size with new videos (Section 5.5.4)?

5.5.1 Experiment Setup

Volumetric Video Source. The volumetric video dataset employed in this evaluation is from the 8i Voxelized Full Bodies version 2 (8iVFB v2) [20]. This dataset contains four distinct video sequences: *longdress*, *loot*, *soldier*, and *red and black*, each consisting of 300 frames. We segment each video sequence into 30 GoFs, with each GoF comprising 10 frames. The original spatial resolution of these videos is $1024 \times 1024 \times 1024$. We provide three tile sizes, $\mathcal{Z} = \{512 \times 512 \times 512, 341 \times 341 \times 341, 256 \times 256 \times 256\}$. That is, the volumetric video frames are evenly partitioned into $2 \times 2 \times 2$, $3 \times 3 \times 3$, and $4 \times 4 \times 4$ tiles. Additionally, five quality levels (point cloud density) are used for tile bitrate allocation $\mathcal{Q} = \{0, 0.25, 0.5, 0.75, 1\}$.

Viewport Source. To simulate user interaction and viewing patterns, we use a viewport dataset [58, 68] consisting of 40 user viewport trajectories across the four videos in 8iVFB v2. Each viewport record contains position dimensions (x, y, z) and rotation dimensions (r_x, r_y, r_z).

Communication Resource. We use the same network traces in Pensieve [69] for RL training and testing. These network traces are collected under diverse scenarios including both static and dynamic conditions (e.g., bus, car, ferry), covering a wide range of real-world network environments. To support volumetric video streaming, we adjust the average bandwidth throughput of the original traces to range between 90-300 Mbps.

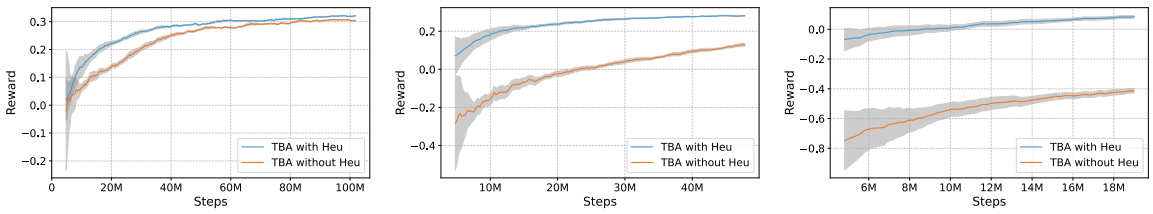
Model and Metric Settings. The states of the TBA agent and TSS agent both

include the past 5 GoF information. The architecture of networks employed in these two agents contains three hidden layers, with 256, 128, and 128 neurons, respectively. The discount factor is set as $\gamma = 0.99$, and the learning rate is a linear function from 10^{-4} to 0. Besides, the parameters of QoE are set as $\omega_1 = 1, \omega_2 = 0.6$.

Training and Testing Data. The TBA and TSS agents are trained using the video *longdress* video sequence, with 30 viewport trajectories and a variety of network traces selected from Pensieve [69]. We first conduct experiments with the same video *longdress* for the remaining 10 viewport trajectories and the rest network traces. To further demonstrate the adaptability of the TBA and TSS agents, we extend the evaluation to three additional videos: *loot*, *red and black*, and *soldier*, under the same experimental conditions.

Baselines. We compare our HRL-based streaming scheme with flexible tile sizes (HRL) against the following four baselines.

- **Pensieve [69].** This is an RL-based 2D video streaming scheme. The scheme does not tile volumetric videos but allocates qualities for the whole volumetric video.
- **ViVo [27].** This scheme uses the fixed tile size and allocates bitrate to tiles based on visibility scores alone. We select the middle tile size $z = 341 \times 341 \times 341$ in this algorithm.
- **ViVo with TSS (ViVo-TSS).** We enable ViVo with flexible tile size by integrating the TSS model with it. The TSS module is dedicatedly trained for ViVo to provide flexible tile size selection.
- **TBA**. This scheme optimizes tile bitrate allocation with the fixed tile size. We set the tile size as $z = 341 \times 341 \times 341$, the same as used in ViVo with TSS.



(a) Tile size = 512^3. (b) Tile size = 341^3. (c) Tile size = 256^3.

Figure 5.13: Training performance comparison between TBA with and without heuristic algorithms.

5.5.2 The Training Performance of Heuristics-Based and Direct TBA Agents

In this section, we evaluate the impact of integrating a heuristic algorithm into the training of the TBA by comparing the performance of a heuristic-based TBA agent and a direct TBA agent. Both agents share the identical network architecture and action space. The observations are the same except for an additional state, the allocation provided by the heuristic algorithm (ViVo in this case), for the heuristics-based TBA agent.

Convergence Speed and Learning Performance. Fig. 5.13 illustrates the evaluation reward during training for two types of TBA agents across different tile sizes. The results demonstrate that it significantly enhances both training performance and convergence speed across all tile sizes by incorporating the heuristic algorithm. Specifically, the final reward is improved from 0.30 to 0.32 with a tile size of 512^3, 0.15 to 0.28 with a tile size of 341^3, -0.36 to 0.11 with a tile size of 256^3. The improvement becomes more obvious as tile size becomes smaller because the smaller tile size leads to a larger and more complex action space. The large action space poses significant challenges for RL to efficiently explore and optimize. Though the heuristic algorithm mitigates this challenge, the large space action still has negative impact on RL performance, especially at tile size of 256^3. As for convergence speed, the heuristic algorithm improves it as expected. For example, the direct TBA agent with a tile size of 512^3 begins to converge

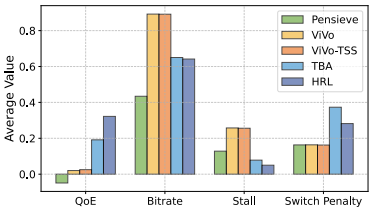


Figure 5.14: Average QoE metrics comparison.

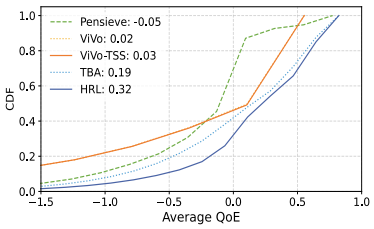


Figure 5.15: CDF of GoF QoE comparison.

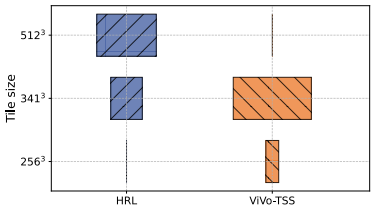


Figure 5.16: Tile selection of different algorithms.

around 50 million steps, while the heuristics-based TBA agent converges much earlier, at approximately 30 million steps.

Long-term Stability and Deployment Considerations. While the proposed HRL framework with heuristics demonstrates improved training efficiency and faster convergence, we also evaluate its long-term stability over extended training steps. Across all tile sizes, the performance of the heuristics-based TBA agent stabilizes after convergence without significant fluctuations, indicating good stability characteristics. From a deployment perspective, the two-stage decision process, which involves sequential inference by the TSS agent followed by the TBA agent, introduces moderate computational overhead. However, both agents employ lightweight neural networks and operate at segment-level granularity rather than per-frame, which helps contain the overall latency. In our following simulation results, the added inference time led to only a slight increase in average streaming delay, which is outweighed by the QoE improvement.

5.5.3 The Performance of HRL-based Volumetric video Streaming Scheme with Flexible Tile Size

In this subsection, we compare the QoE Performance of the HRL approach with baselines. The video resource is *longdress*, watched by 10 users under 142 different network traces.

Overall QoE performance and tile selection behavior. We assess five algorithms by analyzing the QoE metric from different angles. The average values of QoE and its components across diverse users and network traces are shown in Fig. 5.14. Among all algorithms, HRL achieves the highest QoE value, while Pensieve has the worst QoE. The QoE of ViVo-TSS is slightly higher than the one of ViVo, indicating the limited improvement of the TSS module in ViVo. In contrast, the TSS module significantly improves the performance of the TBA agent by 68%, from 0.19 to 0.32.

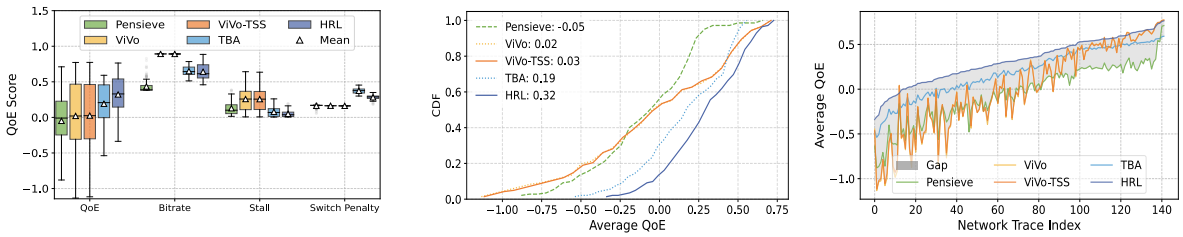
The performance of Pensieve is hindered by unnecessary data transmission such as contents outside FoV and occluded by others. This excessive transmission data decreases the overall bitrate and leads to a high stall. Though ViVo introduces tiling in volumetric video streaming, it allocates bitrate only based on visibility, ignoring the effect of bandwidth and buffer level. Hence, ViVo and ViVo-TSS have the highest average bitrate but the highest stall as well. TBA improves ViVo by considering additional factors such as network conditions and buffer level. Hence, it reduces stalls while at the cost of a slightly lower bitrate and higher switch penalty. HRL compensates for it by adjusting tile size to reduce transmission and computation latency. Moreover, the flexible tile sizes improve average bitrate and reduce quality switch penalty by selecting the most appropriate size under varying situations.

The cumulative distribution function (CDF) of GoF QoE is illustrated in Fig. 5.15. HRL outperforms the baseline algorithms, with a larger proportion of GoFs achieving high QoE. Approximately 80% of GoFs in HRL achieve a positive QoE, which indicates that HRL is able to make an effective decision under most scenarios. TBA ranks second, with around 60% of GoFs attaining positive QoE. While HRL and TBA exhibit similar ratios of GoFs with higher QoE, TBA has a greater proportion of GoFs with lower QoE. It highlights the low adaptability of TBA alone. ViVo and ViVo-TSS have similar GoF QoE distribution, with less than 60% GoFs with a positive QoE. There is a steep improvement

in GoFs with QoE above 0.2, suggesting that these algorithms perform well in some situations, specifically when bandwidth throughput is enough to support high-quality video transmission. Pensieve has a very low proportion of GoFs with high QoE primarily due to the unnecessary content transmission. This inefficiency results in lower bitrate, even under high-bandwidth conditions, further hindering performance.

Fig. 5.16 depicts the tile selection behavior of the TSS module in both HRL and ViVo-TSS algorithms. HRL mainly selects tile sizes between 341^3 and 512^3, while ViVo-TSS prefers smaller tile sizes, typically between 256^3 and 341^3. As discussed in Section 5.5.2, TBA with a tile size of 256^3 has poor performance and hence it leads to infrequent selection in the TSS agent. For ViVo, the imbalance between the high tile quality and low network throughput is the main bottleneck. Therefore, the large tile size such as 512^3 results in higher data transmission redundancy and hence becomes a bad choice for ViVo. Conversely, smaller tile sizes, such as 256^3, occasionally yield better outcomes by mitigating transmission redundancies, especially under constrained network conditions. However, the inefficient tile allocation scheme in ViVo limits the improvement brought by the flexible tile selection.

QoE performance under different network traces. We compare values of QoE and its components of all users under different network traces among five algorithms, as illustrated in Fig. 5.17. As shown in Fig. 5.17(a), the HRL approach has the highest QoE value and smallest QoE range under different network traces, while TBA has a lower QoE and larger range, which demonstrates the effectiveness of flexible tile selection in HRL. ViVo and ViVo-TSS have the largest QoE range, indicating their poor adaptability to fluctuating network conditions. Pensieve has the lowest QoE value but a shorter QoE range. The short range illustrates that RL has the ability to adapt to different situations, while the low QoE indicates that the entire volumetric video streaming significantly decreases QoE due to excessive data transmission.



(a) QoE metric and its components. (b) CDF of video QoE. (c) Video QoE.

Figure 5.17: QoE comparison under different network traces.

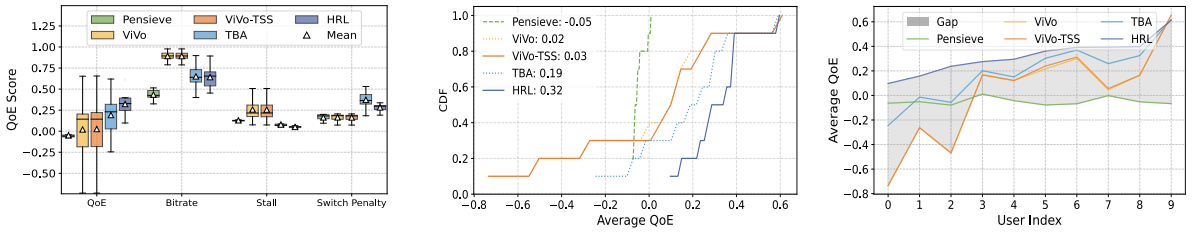
From the perspective of QoE components, HRL has a wider bitrate range than TBA because the TSS module selects dynamic tile sizes, resulting in a more dynamic tile bitrate. ViVo and ViVo-TSS exhibit the highest bitrate as they always select the highest quality, ignoring bandwidth and buffer constraints. Pensieve has the lowest bitrate because it is difficult to allocate a high bitrate for the entire frame due to low bandwidth throughput. The TSS module in the HRL approach significantly reduces the stall by dynamically selecting appropriate tile sizes to match network conditions and user behaviors. Conversely, ViVo and ViVo-TSS experience the highest stall times, as their algorithms fail to account for varying network conditions. Pensieve achieves relatively low stall times thanks to the RL-based algorithm. In terms of quality switches, Pensieve, ViVo, and ViVo-TSS achieve low switch penalties, which are mainly composed of intra-frame switches since the bitrate among tiles is the same. TBA introduces higher quality switches caused by extra inner-frame switches since it allows different tile bitrate allocation. HRL mitigates this issue by leveraging its adaptive tile size selection mechanism, thereby reducing quality switches while maintaining a balance between bitrate and stall time.

The CDF of video QoE under different network traces is shown in Fig. 5.17(b). The HRL approach achieves high QoEs under most situations, while TBA ranks second. The QoE range of ViVo and ViVo-TSS is the largest, indicating their limited adaptability to

varying network conditions. Pensieve has the smallest portion of high QoE performances, suggesting it can not perform well under testing network scenarios. The Fig. 5.17(c) plots the detailed QoE under each network trace. HRL demonstrates the best performance under most network traces, and TBA ranks second in the majority of cases. However, TBA occasionally achieves lower QoE values compared to ViVo and ViVo-TSS due to its conservative approach, which prioritizes playback smoothness rather than high bitrate. Hence, when network throughput is sufficient to support high-quality video all the time, TBA might perform worse than ViVo and ViVo-TSS. Despite occasionally exceptional performance by ViVo and ViVo-TSS, their lack of stability is a significant drawback. While they achieve good results in some cases, they often perform poorly under challenging conditions. Pensieve exhibits the worst performance across most network traces, reinforcing its unsuitability for volumetric video streaming.

QoE performance under different user watching behaviors. In Fig. 5.18, we compare values of QoE and its components of each user across all network traces to analyze the impact of varying user watching behaviors. As shown in Fig. 5.18(a), the HRL approach achieves the highest QoE values and a narrow QoE range, demonstrating its robustness in adapting to diverse user behaviors. In contrast, TBA exhibits lower QoE values with a wider range, highlighting the additional benefits of flexible tile selection integrated into HRL. ViVo and ViVo-TSS show the largest QoE ranges, indicating their limited adaptability to varying user behaviors. Pensieve, while having the lowest QoE, exhibits the smallest QoE range, as it transmits the entire volumetric video without performing content culling. This static transmission approach prevents Pensieve from user behavior variability but results in consistently low QoE due to inefficient resource utilization.

As for QoE components, user watching behaviors significantly influence the allocated video bitrate on algorithms other than ViVo and ViVo-TSS, resulting in a broader range



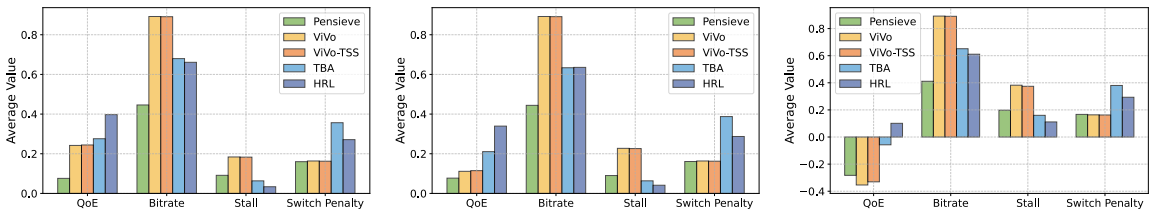
(a) QoE metric and its components. (b) CDF of video QoE. (c) Video QoE.

Figure 5.18: QoE comparison under different users.

of QoE values. The RL-based algorithms have a low and similar video stall, while ViVo and ViVo-TSS lead to high and diverse stall times. HRL reduces the diversity of quality switches of TBA due to flexible tile sizes. The RL-based approaches achieve consistently low stall times, reflecting their ability to maintain smooth playback under diverse user behaviors. In contrast, ViVo and ViVo-TSS result in both high and diverse stall times due to their failure to adapt to network and user variations effectively. TBA introduces a higher switch penalty and HRL reduces it to some extent.

The CDF of video QoE under different user behaviors is illustrated in Fig. 5.18(b). HRL achieves high QoE values for most users, while TBA ranks second. ViVo and ViVo-TSS exhibit the widest QoE range, suggesting their poor adaptability to diverse user behaviors. Pensieve shows a narrow QoE range due to its static, behavior-agnostic approach, but its performance remains limited.

Fig. 5.18(c) provides a detailed view of QoE values for each user. HRL has consistently outperforms other methods across the majority of users, followed by TBA. While ViVo and ViVo-TSS occasionally achieve good performance under specific user behaviors, they often suffer from poor QoE. For example, when users focus on only a small portion of the video, i.e., a limited number of tiles, such as user 9, all tile-based algorithms perform better. ViVo and ViVo-TSS particularly benefit from this behavior, as the reduced tile content requires less bandwidth. However, when users frequently view the entire object in the



(a) Loot. (b) Red and Black. (c) Soldier.

Figure 5.19: QoE comparison of different videos.

video, ViVo and ViVo-TSS experience significantly degraded performance, occasionally performing even worse than Pensieve. This lack of flexibility represents a significant limitation for both ViVo and ViVo-TSS. Pensieve shows similar performance across users, with consistently low QoE, because the user behavior has no influence on the whole frame transmission.

5.5.4 The Performance of HRL-based Scheme with Different Videos

In this subsection, we evaluate the ability of the HRL-based scheme to adjust to new scenarios by conducting experiments with different videos. This evaluation is crucial to understand whether the proposed method can generalize effectively across varying video contents.

Fig. 5.19 shows the values of QoE and its components for 3 different videos, i.e., *loot*, *red and black*, and *soldier*. These sequences are transmitted with the same testing network traces as in the previous evaluations. HRL outperforms other algorithms across all tested video contents and TBA ranks second. This evidences the adaptive capabilities of RL-based algorithms.

Video characteristics have a great influence on user QoE values. For instance, Pensieve has the worst performance with the video *loot* and *red and black* but outperforms

ViVo and ViVo-TSS with the video *soldier*. It happens because video *soldier* has a larger data amount than video *loot* and *red and black*, resulting in lower QoE under identical network conditions. Furthermore, while RL-based algorithms dynamically adjust their bitrate allocation based on video content, ViVo and ViVo-TSS consistently select the highest quality tiles regardless of network or video conditions. Consequently, stall times increase with the data volume of the video sequences, whereas quality switches remain relatively consistent across different videos.

In conclusion, the distinguished performance of HRL across a wide range of videos highlights its adaptability. This adaptability demonstrates the feasibility of deploying a pre-trained model trained on specific video sequences to generalize effectively across other video content, making it a practical solution for volumetric video streaming systems.

5.6 Chapter Summary

This chapter investigates how tile size impacts the trade-off between computing and bandwidth resources in volumetric video streaming. We propose a flexible-tile streaming framework that dynamically adjusts both tile size and bitrate to optimize QoE.

The key contributions of this chapter are as follows:

- We demonstrate that flexible tile size improves culling efficiency and enables better resource adaptation in volumetric video streaming.
- We formulate a joint tile size selection and bitrate allocation problem and address it using a hierarchical reinforcement learning (HRL) framework.
- We design a two-agent architecture: a Tile Size Selection (TSS) agent and a Tile Bitrate Allocation (TBA) agent to manage tile configurations under uncertainty.

- Trace-driven evaluations show that our method significantly improves long-term QoE by increasing video quality and reducing quality switching, with only a slight increase in delay compared to fixed-tile baselines.

These results confirm the effectiveness of dynamic tile configuration in achieving a better balance between visual quality, stability, and system responsiveness under constrained resources.

CHAPTER 6

CONCLUSION AND FUTURE WORK

6.1 Conclusion

This thesis has explored innovative solutions to enhance resource efficiency in video analytics and volumetric video streaming for immersive media applications. It presents three works, each addressing a specific aspect of the challenges.

Address the limited and fixed computing resources in video analytics. In the first work, we introduce Gemini+, a dual-image FPGA-based video analytics pipeline to overcome the limitations of fixed computing resources. This approach configures a CPU-image and a GPU-image within the dual-image FPGA, enabling flexible switching between CPU and GPU resources during runtime. We investigate CPU-GPU resource control and configuration adaptation for real-time video analytics, formulating two critical problems: single-camera and multi-camera dual computing resource control. We analyze the computational complexities of these problems and develop both optimal and sub-optimal algorithms. Simulations demonstrate that flexible CPU-GPU resource

control significantly enhances analytic accuracy compared to fixed CPU-GPU resource systems and GPU-dominant systems. Furthermore, we implement a Gemini+ prototype and conduct a real-world case study, demonstrating its practical feasibility.

Real-time video analytics in domains such as smart city surveillance, autonomous systems, and intelligent manufacturing can greatly benefit from this approach, particularly in scenarios with resource constraints. For instance, edge devices deployed in urban environments or factory floors often operate under limited computational budgets but require consistent analytic performance. By enabling dynamic switching between CPU and GPU resources at runtime, Gemini+ provides a practical and scalable solution for supporting reliable analytics in such constrained and latency-sensitive environments.

Address the limited and dynamic bandwidth resources in volumetric video streaming. In the second work, we introduce layering in volumetric video streaming to enhance prefetching techniques, enabling more efficient utilization of bandwidth resources. We conduct a systematic study on volumetric video streaming, compression, and layer-based prefetching, considering two practical streaming scenarios: allowing or forbidding frame skips. In the frame-skipping scenario, the streaming scheme prioritizes video quality enhancement, while in the non-frame-skipping scenario, the scheme aims to improve QoE including video quality, quality switch, and stall duration. We analyze the computational complexities of these problems and develop efficient algorithms. Simulations show that prefetching-enabled volumetric video streaming significantly improves video quality and reduces skipped frames in the frame-skipping scenario. In the non-frame-skipping scenario, the layer-based prefetching markedly improves QoE and reduces bandwidth wastage. We also implement a layer-based volumetric video streaming framework, validating its effectiveness in real-world conditions.

The proposed methods enhance bandwidth adaptability and maintain consistent

quality, which makes them especially valuable for immersive applications like remote collaboration, virtual meetings, and VR/AR-based education. These scenarios often require high visual quality and low latency, yet must stream over variable and often unreliable network connections. By supporting dynamic prefetching and layered adaptation, the proposed framework helps ensure smooth user experiences even under fluctuating bandwidth conditions. This capability is essential for real-world deployments where quality degradation or playback interruptions can significantly affect usability and user engagement. Furthermore, the layer-based design offers scalability for future high-resolution volumetric content as immersive technologies continue to evolve.

Address the trade-off between limited computing and bandwidth resources in tile-based volumetric video streaming. In the third work, we put emphasis on the importance of tile size in balancing computing and bandwidth resources. We propose a novel approach that considers both tile size selection and bitrate allocation to enhance long-term QoE. We systematically study the impact of tile size, formulating the tile size selection and bitrate allocation problem as a Markov Decision Process. To address uncertain future viewport and bandwidth information, we employ a hierarchical reinforcement learning (HRL) approach to dynamically determine tile size and quality. The HRL agent observes various features and autonomously learns optimal strategies. Trace-driven simulations indicate that flexible-tile volumetric video streaming can improve QoE under different network conditions, video sources, and user behaviors compared to fixed-tile size streaming schemes.

As immersive media applications continue to expand, the ability to adaptively manage both computing and bandwidth resources becomes increasingly critical. This work offers a robust framework for efficient volumetric content delivery in scenarios that demand scalability, adaptability, and personalization. Potential industry applications include virtual events, metaverse platforms, and interactive entertainment experiences, where

users dynamically navigate rich 3D environments and expect consistently high-quality, low-latency visuals. The proposed HRL-based tile size selection mechanism enables fine-grained control over streaming parameters in response to variable user behavior and network conditions, positioning it as a future-ready solution for emerging large-scale, user-centric immersive systems.

6.2 Ethical and Privacy Considerations

As immersive media technologies are widely employed in real-world applications such as smart surveillance, virtual meetings, remote collaboration, and AR/VR environments, it is important to consider the ethical implications and privacy concerns related to the proposed methods in this thesis.

In the first work, a dual-image FPGA-based flexible computing architecture is proposed to enhance video analytics performance. As video analytics often involves the processing of sensitive visual data such as faces, behaviors, or identities, ethical concerns related to surveillance, bias, and data misuse must be carefully addressed. While our technical contributions focus on improving computational efficiency and accuracy, it is crucial to ensure that such systems are deployed within a well-defined ethical framework. This includes obtaining informed consent where applicable, ensuring data anonymization, and incorporating privacy-preserving techniques (e.g., edge processing, on-device analysis) that reduce the exposure of raw visual data to centralized systems.

Across the two studies on volumetric video streaming presented in this thesis, while the emphasis is placed on optimizing resource usage and improving QoE, privacy-related issues also emerge, particularly concerning viewport prediction and user interaction data. To mitigate such concerns, we intentionally designed the system to execute viewport culling and related processing on the client side, thereby avoiding the transmission of

fine-grained user attention or behavioral data to the server. This approach helps preserve user privacy by minimizing the risk of tracking and profiling based on viewing behavior. Nevertheless, as immersive media becomes more personalized and interactive, further approaches such as differential privacy techniques, secure multi-party computation, or federated learning could be integrated to enhance privacy guarantees while enabling user-centric optimizations.

6.3 Future Work

While the thesis has made significant progress on optimizing resource efficiency for video analytics and volumetric video streaming in immersive media applications, several areas remain open for future exploration.

First, the first work on video analytics pipelines with flexible computing resources can be expanded. Currently, the method considers only a single dual-image FPGA. Future research could explore the integration of multiple dual-image FPGAs to further enhance video analytics capabilities. Additionally, a combination of fixed and flexible computing resources could be employed, presenting new challenges in the allocation and scheduling of these heterogeneous computing resources.

This direction is particularly relevant to edge-cloud hybrid computing environments commonly found in smart cities, autonomous infrastructure monitoring, and industrial IoT systems. In these scenarios, scalable and flexible deployment of heterogeneous processing units is crucial for enabling low-latency, high-throughput analytics across geographically distributed nodes. By advancing towards multi-FPGA or hybrid computing frameworks, future systems can better support complex workloads such as real-time video understanding, situational awareness, and predictive maintenance in large-scale deployments.

For the volumetric video streaming studies, the thesis primarily adopted the Google Draco codec due to its fast decoding capabilities. However, Draco lacks inter-frame compression, resulting in suboptimal compression ratios and a reduction in bandwidth utilization. Future work should focus on new streaming schemes based on more advanced compression algorithms.

Potential candidates include emerging geometry-aware and video-based codecs such as MPEG G-PCC and MPEG V-PCC. These codecs offer improved compression efficiency through temporal prediction and inter-frame coding, making them especially suitable for bandwidth-constrained scenarios. However, their current encoding and decoding speeds remain relatively slow, which may limit real-time deployment. With the advancement of hardware acceleration support, these codecs are expected to become more practical for real-time, high-volume applications. Their adoption could significantly enhance streaming performance and visual quality in mobile or resource-limited environments, such as 5G/6G-enabled AR/VR applications, remote assistance systems, and immersive telepresence, where efficient volumetric data transmission is critical for delivering seamless user experiences.

Finally, the thesis focuses solely on the PtCl format of volumetric videos. Future research should explore the computing and bandwidth trade-offs for other volumetric video formats, such as NeRF, light fields and 3DGS, to broaden the applicability and effectiveness of the proposed solutions.

These emerging formats exhibit distinct characteristics in data representation and computational requirements. For instance, NeRF and 3DGS rely on neural rendering and often require intensive real-time computation, whereas light fields emphasize dense view interpolation and demand significant bandwidth for storage and transmission. Investigating the trade-offs between computing and bandwidth for these formats could

reveal new challenges and optimization opportunities, especially for real-time streaming and edge deployment. Extending the proposed techniques to support these formats would significantly enhance the generality and practical value of the thesis contributions in future immersive media systems.

Bibliography

[1] *4dviews - volumetric video capture technology*.
https://www.4dviews.com.

[2] *8i: Stream volumetric video in vr and ar*.
https://8i.com.

[3] *City of auburn toomer's corner webcam 2*.
https://www.youtube.com/watch?v=hMYIc5ZPJL4.

[4] *Draco 3d data compression*.
https://google.github.io/draco/.

[5] *Intel realsense*.
https://www.intelrealsense.com.

[6] *Kinect for windows*.
https://developer.microsoft.com/en-us/windows/kinect.

[7] *Microsoft mixed reality capture studio*.
https://www.microsoft.com/en-us/mixed-reality/capture-studios.

[8] R. Al Amin, M. Hasan, V. Wiese, and R. Obermaisser, *Fpga-based real-time object detection and classification system using yolo for edge computing*, in IEEE Access, 2024.

[9] M. ALI, A. ANJUM, O. RANA, A. R. ZAMANI, D. BALOUEK-THOMERT, AND M. PARASHAR, *Res: Real-time video stream analytics using edge enhanced clouds*, in IEEE Transactions on Cloud Computing, vol. 10, 2020, pp. 792–804.

[10] K. APICHARTTRISORN, X. RAN, J. CHEN, S. V. KRISHNAMURTHY, AND A. K. ROY-CHOWDHURY, *Frugal following: power thrifty object detection and tracking for mobile augmented reality*, in Proc. ACM SenSys, 2019, pp. 96–109.

[11] U. ASSARSSON AND T. MOLLER, *Optimized view frustum culling algorithms for bounding boxes*, in Journal of graphics tools, vol. 5, 2000, pp. 9–22.

[12] C. CANEL, T. KIM, G. ZHOU, C. LI, H. LIM, D. ANDERSEN, M. KAMINSKY, AND S. R. DULLOOR, *Scaling video analytics on constrained edge nodes*, in Proc. MLSys, 2019.

[13] N. CARION, F. MASSA, G. SYNNAEVE, N. USUNIER, A. KIRILLOV, AND S. ZAGORUYKO, *End-to-end object detection with transformers*, in Computer Vision - ECCV, 2020, pp. 213–229.

[14] S. O. E. CHARLES LOOP, QIN CAI AND P. A. CHOU, *Microsoft voxelized upper bodies - a voxelized point cloud dataset*, in ISO/IEC JTC1/SC29 Joint WG11/WG1 (MPEG/JPEG) input document m38673/M72012, 2016.

[15] J. CHEN, J. CHEN, X. HUANG, Q. LIU, M. SONG, AND H. ZHANG, *Edge-assisted adaptive heterogeneous resource allocation optimization for large-scale live video analytics*, Electronics Letters, 61 (2025), p. e70287.

[16] A. COLLET, M. CHUANG, P. SWEENEY, D. GILLETT, D. EVSEEV, D. CALABRESE, H. HOPPE, A. KIRK, AND S. SULLIVAN, *High-quality streamable free-viewpoint video*, in ACM Transactions on Graphics, vol. 34, 2015, pp. 69:1–69:13.

[17] M. DANTONE, L. BOSSARD, T. QUACK, AND L. VAN GOOL, *Augmented faces*, in Proc. IEEE ICCV workshops, 2011, pp. 24–31.

[18] J. DEAN, *Recent advances in artificial intelligence via machine learning and the implications for computer system design*, in Proc. IEEE HCS, 2017.

[19] A. ELGABLI, V. AGGARWAL, S. HAO, F. QIAN, AND S. SEN, *Lbp: Robust rate adaptation algorithm for svc video streaming*, in IEEE/ACM Transactions on Networking, vol. 26, 2018, pp. 1633–1645.

[20] T. M. EUGENE D'EON, BOB HARRISON AND P. A. CHOU, *8i voxelized full bodies - a voxelized point cloud dataset*, in ISO/IEC JTC1/SC29 Joint WG11/WG1 (MPEG/JPEG) input document WG11M40059/WG1M74006, 2017.

[21] J. D. FOLEY, *Computer graphics: principles and practice*, vol. 12110, 1996.

[22] A. GALANOPOULOS, J. A. AYALA-ROMERO, D. J. LEITH, AND G. IOSIFIDIS, *Automl for video analytics with edge computing*, in Proc. IEEE INFOCOM 2021, 2021, pp. 1–10.

[23] S. GANDHARE AND B. KARTHIKEYAN, *Survey on fpga architecture and recent applications*, in 2019 International Conference on Vision Towards Emerging Trends in Communication and Networking (ViTECoN), IEEE, 2019, pp. 1–4.

[24] Y. GAO, P. ZHOU, Z. LIU, B. HAN, AND P. HUI, *Fras: Federated reinforcement learning empowered adaptive point cloud video streaming*, in arXiv preprint arXiv:2207.07394, 2022.

[25] B. GARCIA-GARCIA, T. BOUWMANS, AND A. J. R. SILVA, *Background subtraction in real applications: Challenges, current models and future directions*, in Computer Science Review, 2020, pp. 100–204.

[26] R. B. GIRSHICK, J. DONAHUE, T. DARRELL, AND J. MALIK, *Rich feature hierarchies for accurate object detection and semantic segmentation*, in Proc. IEEE CVPR, 2014, pp. 580–587.

[27] B. HAN, Y. LIU, AND F. QIAN, *Vivo: visibility-aware mobile volumetric video streaming*, in Proc. ACM MobiCom, 2020.

[28] S. HAN, H. MAO, AND W. DALLY, *Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding*, in Proc. ICLR, 2016.

[29] M. HANYAO, Y. JIN, Z. QIAN, S. ZHANG, AND S. LU, *Edge-assisted online on-device object detection for real-time video analytics*, in Proc. IEEE INFOCOM, 2021, pp. 1–10.

[30] K. HE, X. ZHANG, S. REN, AND J. SUN, *Deep residual learning for image recognition*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.

[31] Y. HE, P. YANG, T. QIN, AND N. ZHANG, *End-edge coordinated joint encoding and neural enhancement for low-light video analytics*, in Proc. IEEE GLOBECOM, 2023, pp. 7363–7368.

[32] H. HEIDARIRAD AND M. WANG, *Vv-dash: A framework for volumetric video dash streaming*, in Proceedings of the 16th ACM Multimedia Systems Conference, 2025, pp. 256–262.

[33] M. HOSSEINI AND C. TIMMERER, *Dynamic adaptive point cloud streaming*, in Proc. ACM PV, 2018, pp. 25–30.

[34] A. G. HOWARD, M. ZHU, B. CHEN, D. KALENICHENKO, W. WANG, T. WEYAND, M. ANDREETTO, AND H. ADAM, *Mobilenets: Efficient convolutional neural*

networks for mobile vision applications, in arXiv preprint arXiv:1704.04861, 2017.

[35] C. HU, W. BAO, D. WANG, AND F. LIU, *Dynamic adaptive dnn surgery for inference acceleration on the edge*, in Proc. IEEE INFOCOM, 2019, pp. 1423–1431.

[36] C. HU, R. LU, Q. SANG, H. LIANG, D. WANG, D. CHENG, J. ZHANG, Q. LI, AND J. PENG, *An edge-side real-time video analytics system with dual computing resource control*, in IEEE Transactions on Computers, vol. 72, 2023, pp. 3399–3415.

[37] K. HU, Y. JIN, H. YANG, J. LIU, AND F. WANG, *Fsvvd: A dataset of full scene volumetric video*, in Proc. ACM MMSys, 2023, pp. 410–415.

[38] K. HU, H. YANG, Y. JIN, J. LIU, Y. CHEN, M. ZHANG, AND F. WANG, *Understanding user behavior in volumetric video watching: Dataset, analysis and prediction*, in Proc. ACM MM, 2023, pp. 1108–1116.

[39] T.-Y. HUANG, R. JOHARI, N. MCKEOWN, M. TRUNNELL, AND M. WATSON, *A buffer-based approach to rate adaptation: Evidence from a large video streaming service*, in Proceedings of the 2014 ACM conference on SIGCOMM, 2014, pp. 187–198.

[40] M. HUTSEBAUT-BUYSSE, K. METS, AND S. LATRÉ, *Hierarchical reinforcement learning: A survey and open research challenges*, in Machine Learning and Knowledge Extraction, vol. 4, 2022, pp. 172–221.

[41] R. HUYSEGEMS, B. DE VLEESCHAUWER, T. WU, AND W. VAN LEEKWIJCK, *Svc-based http adaptive streaming*, in Bell Labs Technical Journal, vol. 16, 2012, pp. 25–41.

[42] F. N. IANDOLA, M. MOSKEWICZ, K. ASHRAF, S. HAN, W. DALLY, AND K. KEUTZER, *Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size*, in arXiv preprint arXiv:1602.07360, 2016.

[43] K. JAIN, K. S. ADAPA, K. GROVER, R. K. SARVADEVABHATLA, AND S. PURINI, *A cloud-fog architecture for video analytics on large scale camera networks using semantic scene analysis*, in Proc. IEEE/ACM CCGrid, 2023, pp. 513–523.

[44] E. S. JANG, M. PREDA, K. MAMMOU, A. M. TOURAPIS, J. KIM, D. B. GRAZIOSI, S. RHYU, AND M. BUDAGAVI, *Video-based point-cloud-compression standard in mpeg: From evidence collection to committee draft [standards in a nutshell]*, in IEEE Signal Processing Magazine, vol. 36, 2019, pp. 118–123.

[45] H. JOO, T. SIMON, X. LI, H. LIU, L. TAN, L. GUI, S. BANERJEE, T. S. GODISART, B. NABBE, I. MATTHEWS, T. KANADE, S. NOBUHARA, AND Y. SHEIKH, *Panoptic studio: A massively multiview system for social interaction capture*, in IEEE Transactions on Pattern Analysis and Machine Intelligence, 2017.

[46] D. KANG, J. EMMONS, F. ABUZAID, P. BAILIS, AND M. ZAHARIA, *Noscope: Optimizing deep cnn-based queries over video streams at scale*, in Proc. VLDB Endow., vol. 10, 2017, pp. 1586–1597.

[47] W. KANG, K. LEE, J. LEE, I. SHIN, AND H. S. CHWA, *Lalarand: Flexible layer-by-layer cpu/gpu scheduling for real-time dnn tasks*, in Proc. IEEE RTSS, 2021, pp. 329–341.

[48] Y. KANG, J. HAUSWALD, C. GAO, A. ROVINSKI, T. N. MUDGE, J. MARS, AND L. TANG, *Neurosurgeon: Collaborative intelligence between the cloud and mobile edge*, in Proc. ACM ASPLOS, 2017.

[49] S. KHULLER, A. MOSS, AND J. S. NAOR, *The budgeted maximum coverage problem*, in Information processing letters, vol. 70, 1999, pp. 39–45.

[50] A. KRAUSE AND C. GUESTRIN, *A note on the budgeted maximization of submodular functions*, 2005.

[51] A. KRIZHEVSKY, I. SUTSKEVER, AND G. E. HINTON, *Imagenet classification with deep convolutional neural networks*, in Commun. ACM, vol. 60, 2017, pp. 84 – 90.

[52] T. D. KULKARNI, K. NARASIMHAN, A. SAEEDI, AND J. TENENBAUM, *Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation*, in Advances in Neural Information Processing Systems 29 (NeurIPS 2016), 2016, pp. 3675–3683.

[53] P. KUMARI, P. CHAURASIA, AND P. KUMAR, *A survey on noise reduction in images*, in Proc. IJCA NSFTICE, 2015, pp. 5–9.

[54] K. LEE, J. YI, Y. LEE, S. CHOI, AND Y. M. KIM, *Groot: a real-time streaming system of high-fidelity volumetric videos*, in Proc. ACM MobiCom, 2020.

[55] H. LI, C. HU, J. JIANG, Z. WANG, Y. WEN, AND W. ZHU, *Jalad: Joint accuracy- and latency-aware deep structure decoupling for edge-cloud execution*, in Proc. IEEE ICPADS, 2018, pp. 671–678.

[56] H. LI, X. LI, Q. FAN, Q. HE, X. WANG, AND V. C. M. LEUNG, *Distributed dnn inference with fine-grained model partitioning in mobile edge computing networks*, IEEE Transactions on Mobile Computing, 23 (2024), pp. 9060–9074.

[57] J. LI, H. WANG, Z. LIU, P. ZHOU, X. CHEN, Q. LI, AND R. HONG, *Towards optimal real-time volumetric video streaming: A rolling optimization and deep*

reinforcement learning based approach, in IEEE Transactions on Circuits and Systems for Video Technology, 2023.

[58] J. LI, X. WANG, Z. LIU, AND Q. LI, *A qoe model in point cloud video streaming*, in arXiv preprint arXiv:2111.02985, 2021.

[59] J. LI, C. ZHANG, Z. LIU, R. HONG, AND H. HU, *Optimal volumetric video streaming with hybrid saliency based tiling*, in IEEE Trans. Multimedia, vol. 25, 2022, pp. 2939–2953.

[60] J. LI, C. ZHANG, Z. LIU, W. SUN, AND Q. LI, *Joint communication and computational resource allocation for qoe-driven point cloud video streaming*, in Proc. IEEE ICC, 2020, pp. 1–6.

[61] R. LI, D. SU, AND H. KAN, *A fast, scalable, and energy-efficient edge acceleration architecture based on fpga cluster*, in Proc. ACM CoNEXT, 2021, pp. 479–480.

[62] Y. LI, A. PADMANABHAN, P. ZHAO, Y. WANG, G. H. XU, AND R. NETRAVALI, *Reducto: On-camera filtering for resource-efficient real-time video analytics*, in Proc. ACM SIGCOMM, 2020.

[63] C.-Y. LIN, T.-C. WANG, K.-C. CHEN, B.-Y. LEE, AND J.-J. KUO, *Distributed deep neural network deployment for smart devices from the edge to the cloud*, in Proc. ACM PERSIST-IoT, 2019.

[64] H. LIN AND X. CHEN, *Transformer-driven multi-agent deep reinforcement learning based point cloud video transmissions*, in Proc. ACM AIIOT, 2022, pp. 25–30.

[65] T.-Y. LIN, P. GOYAL, R. GIRSHICK, K. HE, AND P. DOLLÁR, *Focal loss for dense object detection*, in Proc. IEEE ICCV, 2017, pp. 2980–2988.

[66] J. LIU, B. ZHU, F. WANG, Y. JIN, W. ZHANG, Z. XU, AND S. CUI, *Cav3: Cache-assisted viewport adaptive volumetric video streaming*, in Proc. IEEE VR, 2023, pp. 173–183.

[67] M. LIU, X. DING, AND W. DU, *Continuous, real-time object detection on mobile devices without offloading*, in Proc. IEEE ICDCS, 2020, pp. 976–986.

[68] Z. LIU, Q. LI, X. CHEN, C. WU, S. ISHIHARA, J. LI, AND Y. JI, *Point cloud video streaming: Challenges and solutions*, in IEEE Network, vol. 35, 2021, pp. 202–209.

[69] H. MAO, R. NETRAVALI, AND M. ALIZADEH, *Neural adaptive video streaming with pensieve*, in Proceedings of the ACM SIGCOMM 2017 Conference, 2017, pp. 197–210.

[70] T. MOHAMMED, C. JOE-WONG, R. BABBAR, AND M. D. FRANCESCO, *Distributed inference acceleration with adaptive dnn partitioning and offloading*, in Proc. IEEE INFOCOM, 2020, pp. 854–863.

[71] A. MUSCOLONI AND S. MATTOCCIA, *Real-time tracking with an embedded 3d camera with fpga processing*, in Proc. IEEE IC3D, 2014, pp. 1–7.

[72] S. OH, A. HOOGS, A. PERERA, N. P. CUNTOOR, C.-C. CHEN, J. T. LEE, S. MUKHERJEE, J. AGGARWAL, H. LEE, L. DAVIS, E. SWEARS, X. WANG, Q. JI, K. REDDY, M. SHAH, C. VONDRICK, H. PIRSIAVASH, D. RAMANAN, J. YUEN, A. TORRALBA, B. SONG, A. FONG, A. ROY-CHOWDHURY, AND M. DESAI, *A large-scale benchmark dataset for event recognition in surveillance video*, in Proc. IEEE CVPR, 2011, pp. 3153–3160.

[73] I. PANTAZOPOULOS AND S. TZAFESTAS, *Occlusion culling algorithms: A comprehensive survey*, in Journal of Intelligent and Robotic Systems, vol. 35, 2002, pp. 123–156.

[74] A. PARASHAR, M. RHU, A. MUKKARA, A. PUGLIELLI, R. VENKATESAN, B. KHAILANY, J. EMER, S. W. KECKLER, AND W. J. DALLY, *Scnn: An accelerator for compressed-sparse convolutional neural networks*, in ACM SIGARCH computer architecture news, 2017, pp. 27–40.

[75] J. PARK, P. A. CHOU, AND J.-N. HWANG, *Volumetric media streaming for augmented reality*, in Proc. IEEE GLOBECOM, 2019.

[76] P. PARSANIA, P. V. VIRPARIA, ET AL., *A review: Image interpolation techniques for image scaling*, in International Journal of Innovative Research in Computer and Communication Engineering, 2014, pp. 7409–7414.

[77] S. PATERIA, B. SUBAGDJA, A.-H. TAN, AND C. QUEK, *Hierarchical reinforcement learning: A comprehensive survey*, in ACM Computing Surveys, vol. 54, 2021, pp. 1–35.

[78] M. QASAIMEH, K. DENOLF, J. LO, K. VISSERS, J. ZAMBRENO, AND P. H. JONES, *Comparing energy efficiency of cpu, gpu and fpga implementations for vision kernels*, in IEEE ICESS, 2019, pp. 1–8.

[79] Y. QI, Z. YANG, W. SUN, M. LOU, J. LIAN, W. ZHAO, X. DENG, AND Y. MA, *A comprehensive overview of image enhancement techniques*, in Archives of Computational Methods in Engineering, 2021, pp. 1–25.

[80] D. RACA, D. LEAHY, C. J. SREENAN, AND J. J. QUINLAN, *Beyond throughput, the next generation: a 5g dataset with channel and context metrics*, in Proc. ACM MMSys, 2020.

[81] X. RAN, H. CHEN, X.-D. ZHU, Z. LIU, AND J. CHEN, *Deepdecision: A mobile deep learning framework for edge video analytics*, in Proc. IEEE INFOCOM, 2018, pp. 1421–1429.

[82] M. RASTEGARI, V. ORDONEZ, J. REDMON, AND A. FARHADI, *Xnor-net: Imagenet classification using binary convolutional neural networks*, in Proc. ECCV, 2016, pp. 525–542.

[83] J. B. RAWLINGS, *Tutorial overview of model predictive control*, in IEEE control systems magazine, vol. 20, 2000, pp. 38–52.

[84] J. REDMON, S. DIVVALA, R. B. GIRSHICK, AND A. FARHADI, *You only look once: Unified, real-time object detection*, in Proc. IEEE CVPR, 2016, pp. 779–788.

[85] S. REN, K. HE, R. B. GIRSHICK, AND J. SUN, *Faster r-cnn: Towards real-time object detection with region proposal networks*, in IEEE Trans. Pattern Anal. Mach. Intell., vol. 39, 2017, pp. 1137–1149.

[86] J. SCHULMAN, P. MORITZ, S. LEVINE, M. JORDAN, AND P. ABBEEL, *High-dimensional continuous control using generalized advantage estimation*, in arXiv preprint arXiv:1506.02438, 2015.

[87] J. SCHULMAN, F. WOLSKI, P. DHARIWAL, A. RADFORD, AND O. KLIMOV, *Proximal policy optimization algorithms*, in arXiv preprint arXiv:1707.06347, 2017.

[88] S. SCHWARZ, M. PREDA, V. BARONCINI, M. BUDAGAVI, P. CESAR, P. A. CHOU, R. A. COHEN, M. KRIVOKUfÁA, S. LASSERRE, Z. LI, J. LLACH, K. MAMMOU, R. MEKURIA, O. NAKAGAMI, E. SIAHAAN, A. TABATABAI, A. M. TOURAPIS, AND V. ZAKHARCHENKO, *Emerging mpeg standards for point cloud compression*, in IEEE Journal on Emerging and Selected Topics in Circuits and Systems, vol. 9, 2019, pp. 133–148.

[89] J. SHI, L. PU, X. YUAN, Q. GONG, AND J. XU, *Sophon: Super-resolution enhanced 360 video streaming with visual saliency-aware prefetch*, in Proc. ACM MM, 2022, pp. 3124–3133.

[90] Y. SHI AND S. LICHMAN, *Smart cameras: a review*, in Proc. Asia-Pacific Workshop on Visual Information, 2005, pp. 95–100.

[91] K. SPITERI, R. SITARAMAN, AND D. SPARACIO, *From theory to practice: Improving bitrate adaptation in the dash reference player*, in ACM Transactions on Multimedia Computing, Communications, and Applications, vol. 15, 2019, pp. 1–29.

[92] K. SPITERI, R. URGAONKAR, AND R. K. SITARAMAN, *Bola: Near-optimal bitrate adaptation for online videos*, in IEEE/ACM Transactions on Networking, vol. 28, 2020, pp. 1698–1711.

[93] T. STOCKHAMMER, *Dynamic adaptive streaming over http– standards and design principles*, in Proc. ACM MMSys, 2011, pp. 133–144.

[94] Y.-C. SUN, Y. SHI, C.-T. LEE, M. ZHU, W. T. OOI, Y. LIU, C.-Y. HUANG, AND C.-H. HSU, *Lts: A dash streaming system for dynamic multi-layer 3d gaussian splatting scenes*, in Proceedings of the 16th ACM Multimedia Systems Conference, 2025, pp. 136–147.

[95] H. SUZUKI, *A generalized knapsack problem with variable coefficients*, in Mathematical Programming, vol. 15, 1978, pp. 162–176.

[96] C. SZEGEDY, W. LIU, Y. JIA, P. SERMANET, S. E. REED, D. ANGUELOV, D. ERHAN, V. VANHOUCKE, AND A. RABINOVICH, *Going deeper with convolutions*, in Proc. IEEE CVPR, 2015, pp. 1–9.

[97] M. TAN, R. PANG, AND Q. V. LE, *Efficientdet: Scalable and efficient object detection*, in Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2020, pp. 10781–10790.

[98] X. TANG AND Z. FU, *Cpu‚Äìgpu utilization aware energy-efficient scheduling algorithm on heterogeneous computing systems*, in IEEE Access, vol. 8, 2020, pp. 58948–58958.

[99] S. TEERAPITTAYANON, B. MCDANEL, AND H. T. KUNG, *Distributed deep neural networks over the cloud, the edge and end devices*, in Proc. IEEE ICDCS, 2017, pp. 328–339.

[100] D. TIAN, H. OCHIMIZU, C. FENG, R. COHEN, AND A. VETRO, *Geometric distortion metrics for point cloud compression*, in 2017 IEEE International Conference on Image Processing (ICIP), IEEE, 2017, pp. 3460–3464.

[101] T. TIANXIANG AND C. GUOHONG, *Fastva: Deep learning video analytics through edge processing and npu in mobile*, in Proc. IEEE INFOCOM, 2020, pp. 1947–1956.

[102] J. VAN DER HOOFT, T. WAUTERS, F. DE TURCK, C. TIMMERER, AND H. HELLWAGNER, *Towards 6dof http adaptive streaming through point cloud compression*, in Proc. ACM MM, 2019, pp. 2405–2413.

[103] G. V. D. WAL, D. C. ZHANG, I. KANDASWAMY, J. MARAKOWITZ, K. KAIGHN, J. ZHANG, AND S. CHAI, *Fpga acceleration for feature based processing applications*, in Proc. IEEE CVPRW, 2015, pp. 42–47.

[104] C. WANG, S. ZHANG, Y. CHEN, Z. QIAN, J. WU, AND M. XIAO, *Joint configuration adaptation and bandwidth allocation for edge-based real-time video analytics*, in Proc. IEEE INFOCOM, 2020, pp. 257–266.

[105] L. WANG, C. LI, W. DAI, S. LI, J. ZOU, AND H. XIONG, *Qoe-driven adaptive streaming for point clouds*, in IEEE Transactions on Multimedia, 2022.

[106] S. WANG, C. ZHANG, Y. SHU, AND Y. LIU, *Live video analytics with fpga-based smart cameras*, in Proc. ACM HotEdgeVideo, 2019, pp. 9–14.

[107] S. WANG, C. ZHANG, Y. SHU, AND Y. LIU, *Live video analytics with fpga-based smart cameras*, in HotEdgeVideo, 2019, pp. 9–14.

[108] W. WANG, L. MI, S. CEN, H. DAI, Y. LI, X. FU, AND Y. LIU, *Region-based content enhancement for Efficient video analytics at the edge*, in 22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25), Philadelphia, PA, Apr. 2025, USENIX Association, pp. 613–633.

[109] Y. WANG, H. KAN, D. SU, Y. SHEN, W. LIU, AND M. OU, *Energy efficient computing offloading mechanism based on fpga cluster for edge cloud*, in Proc. ACM EITCE, 2020, pp. 1120–1125.

[110] Y. WANG, G.-Y. WEI, AND D. BROOKS, *Benchmarking tpu, gpu, and cpu platforms for deep learning*, in arXiv preprint arXiv:1907.10701, 2019.

[111] Y. WANG, J. XU, Y. HAN, H. LI, AND X. LI, *Deepburning: Automatic generation of fpga-based learning accelerators for the neural network family*, in Proc. ACM/EDAC/IEEE DAC, 2016, pp. 1–6.

[112] Z. WANG, R. ZHANG, S. ZHANG, W. CHENG, W. WANG, AND Y. CUI, *Edge-assisted adaptive configuration for serverless-based video analytics*, IEEE Transactions on Networking, 33 (2025), pp. 1144–1159.

[113] Y. XIANG AND H. KIM, *Pipelined data-parallel cpu/gpu scheduling for multi-dnn real-time inference*, in Proc. IEEE RTSS, 2019, pp. 392–405.

[114] Y. XIE, Y. ZHANG, AND T. LIN, *Deep curriculum reinforcement learning for adaptive 360° video streaming with two-stage training*, in IEEE Transactions on Broadcasting, 2023.

[115] R. XUKAN, C. HAOLIANZ, X. ZHU, L. ZHENMING, AND C. JIASI, *Deepdecision: A mobile deep learning framework for edge video analytics*, in Proc. IEEE INFOCOM, 2018, pp. 1421–1429.

[116] Y. YAN, S. ZHANG, X. WANG, N. CHEN, Y. CHEN, Y. LIANG, M. XIAO, AND S. LU, *Visflow: Adaptive content-aware video analytics on collaborative cameras*, in IEEE INFOCOM 2024 - IEEE Conference on Computer Communications, Vancouver, BC, Canada, May 20-23, 2024, 2024, pp. 2019–2028.

[117] D. YIN, J. SHI, M. ZHANG, Z. HUANG, J. LIU, AND F. DONG, *Fsvfg: Towards immersive full-scene volumetric video streaming with adaptive feature grid*, in Proceedings of the 32nd ACM International Conference on Multimedia, 2024, pp. 11089–11098.

[118] X. YIN, A. JINDAL, V. SEKAR, AND B. SINOPOLI, *A control-theoretic approach for dynamic adaptive video streaming over http*, in Proc. ACM SIGCOMM, 2015, pp. 325–338.

[119] C. YU, M. WANG, S. CHEN, W. WANG, W. FANG, Y. CHEN, AND N. N XIONG, *Efficient npu–gpu scheduling for real-time deep learning inference on mobile devices*, Journal of Real-Time Image Processing, 22 (2025), pp. 1–13.

[120] A. ZHANG, C. WANG, B. HAN, AND F. QIAN, *Yuzu: Neural-enhanced volumetric video streaming*, in Proc. USENIX NSDI, 2022.

[121] A. ZHANG, C. WANG, B. HAN, AND F. QIAN, *Yuzu: Neural-enhanced volumetric video streaming*, in Proc. USENIX NSDI, 2022.

[122] C. ZHANG, Q. CAO, H. JIANG, W. ZHANG, J. LI, AND J. YAO, *Ffs-va: A fast filtering system for large-scale video analytics*, in Proc. ACM ICPP, 2018, pp. 1–10.

[123] C. ZHANG, Q. CAO, H. JIANG, W. ZHANG, J. LI, AND J. YAO, *A fast filtering mechanism to improve efficiency of large-scale video analytics*, in IEEE Transactions on Computers, vol. 69, 2020, pp. 914–928.

[124] C. ZHANG, Y. CAO, Z. LIU, R. YIN, Y. ZHU, AND X. CHEN, *Trans-rl: A prediction-control approach for qoe-aware point cloud video streaming*, in Proc. IEEE GLOBECOM, 2022, pp. 1899–1904.

[125] D. ZHANG, P. ZHOU, B. HAN, AND P. PATHAK, *M5: Facilitating multi-user volumetric content delivery with multi-lobe multicast over mmwave*, in Proc. ACM Sensys, 2022.

[126] M. ZHANG, F. WANG, AND J. LIU, *Casva: Configuration-adaptive streaming for live video analytics*, in Proc. IEEE INFOCOM, 2022, pp. 2168–2177.

[127] R. ZHANG, J. LIU, F. LIU, T. HUANG, Q. TANG, S. WANG, AND F. R. YU, *Buffer-aware virtual reality video streaming with personalized and private viewport prediction*, in IEEE Journal on Selected Areas in Communications, vol. 40, 2021, pp. 694–709.

[128] G. ZHOU, Z. LUO, M. HU, AND D. WU, *Presr: Neural-enhanced adaptive streaming of vbr-encoded videos with selective prefetching*, in IEEE Transactions on Broadcasting, vol. 69, 2022, pp. 49–61.

[129] A. ZHU, S. ZHANG, X. SHI, K. CHENG, H. SUN, AND S. LU, *Crucio: End-to-end coordinated spatio-temporal redundancy elimination for fast video analytics*, in IEEE INFOCOM 2024 - IEEE Conference on Computer Communications, 2024, pp. 1191–1200.