ANALYSIS AND ENHANCEMENT OF RESOURCE-HUNGRY APPLICATIONS

by

Siqi Huang

A dissertation submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in
Electrical Engineering

Charlotte

2022

Approved by:

Dr. Jiang Xie

Dr. Tao Han

Dr. Pu Wang

Dr. Aidong Lu

ABSTRACT

SIQI HUANG. Analysis and Enhancement of Resource-Hungry Applications.
(Under the direction of DR. JIANG XIE)

Resource-hungry applications play a very important role in people's daily lives, such as real-time video streaming applications and mobile augmented reality applications. However, there are several challenges to satisfy the user Quality-of-Experience (QoE) requirements of resource-hungry applications. First, these applications usually require a vast amount of network bandwidth resources to support the data communication of different functionalities. However, only limited network bandwidth resources can be assigned to these applications which leads to long network latency and poor user QoE. In addition, artificial intelligent (AI) and machine learning (ML) models are widely adopted in these applications which significantly increases the computation complexity of these applications. Because of the limited computing resource on mobile devices, computation-intensive tasks are offloaded to edge servers located at the edge of the core network. However, additional network latency and bandwidth usage are introduced which may degrade user QoE. Moreover, base stations (BSs) and edge servers may be densely deployed to provide high network capacity, thus resulting in ultra-dense wireless networks. However, a major challenge of the ultra-dense wireless network is the increased complexity of networking mechanisms.

In this research, the characteristics of popular resource-hungry applications are first analyzed. Then, based on the analyzed characteristics, we propose several specifically designed algorithms to enhance the performance of each resource-hungry application. For real-time video streaming applications, a configuration adaptive video encoding scheme is proposed to improve the video visual quality of existing real-time video streaming applications. For ultra-high-definition (UHD) video delivery applications, we propose a cloud computing based deep compression framework named Pearl, which

utilizes the power of deep learning and cloud computing to compress UHD videos and reduce the high network bandwidth resource usage of UHD video delivery. For mobile augmented reality (MAR) applications, a fast model updating algorithm and a smart task allocation decision algorithm are proposed to overcome the challenges brought by the high computation complexity and latency-sensitive tasks in MAR applications. Finally, a real-time network optimization algorithm is proposed to address the high complexity problem of networking design in ultra-dense wireless networks.

In our proposed algorithms, deep learning techniques (e.g., neural networks, reinforcement learning, and incremental learning) and machine learning algorithms (e.g., clustering) are adopted. This research will provide important insights for the design of AI-powered optimization for resource-hungry applications and network management.

ACKNOWLEDGEMENTS

Foremost, I would like to express my greatest and deepest gratitude to my advisor, Professor Dr. Jiang Xie, for her continuous support, guidance, and supervision throughout these years. She has patiently spent countless hours to discuss and refine my cluttered research ideas and helped me to become an independent researcher. I could not have imagined having a better advisor than her.

Besides my advisor, I would like to thank my co-advisor Dr. Tao Han, and Ph.D. committee members: Dr. Aidong Lu, Dr. Pu Wang, and Dr. Yu Wang for their time and valuable advice. In addition, I appreciate the GASP grant from UNC-Charlotte and Research Assistantships from National Science Foundation (NSF) as the financial assistance for this work.

In the end, I would like to thank my family and friends. Their wishes and supports made the journey easier. Special thanks to my sister, Dr. Xueqing Huang. She has been extremely supportive.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

1.1 Resource-hungry Applications

The mobile network data are growing explosively with the proliferation of mobile devices and resource-hungry applications, and have led to a continuous surge in capacity demand across mobile networks [1]. The bigger data transmission volume and the higher low-latency requirement are the two main characteristics of resource-hungry applications. On the other hand, to support advanced functions for mobile users, high computation complexity artificial intelligent (AI) and machine learning (ML) models are widely adopted in resource-hungry applications. High computation demand is another characteristic of resource-hungry applications. In this chapter, several popular resource-hungry applications are introduced.

1.1.1 Ultra-high-definition Video Delivery Applications

Ultra-high-definition (UHD) videos (4K and 8K) are enjoying increased popularity in people's daily lives because of the better visual experience compared with HD videos. However, UHD videos bring much higher pressure on video data transmission and storage because the data size of 4k and 8k resolution is 4 times and 16 times larger than 2k resolution for a video, respectively.

Video delivery has been extensively studied. Content Distributed Networks (CDNs) [2, 3] are the main tools that content providers like Google and Netflix use to improve the performance of video delivery. As shown in Figure 1.1(a), a CDN consists of a group of servers that are placed across the world. These servers pull the contents from the origin server and cache a copy of them allowing visitors to retrieve the content from the nearest server. CDNs serve a large portion of the video deliveries

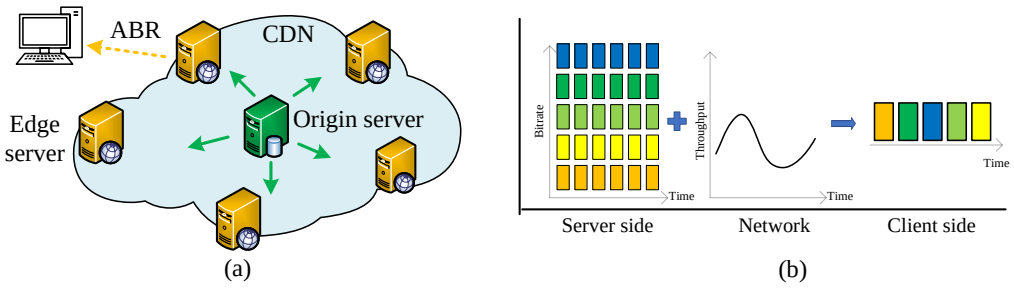


Figure 1.1: The content distributed network system (a) and the adaptive bitrate selection algorithm (b).

across the Internet. One of the main challenges of CDNs is the limited storage of the distributed servers, which leads to cached videos on the distributed servers frequently being replaced [4]. This shortage will be amplified with the increasing amount of UHD videos in CDNs.

Adaptive bitrate (ABR) algorithms [5, 6] are widely used to optimize video transmission. As shown in Figure 1.1(b), a video is encoded into multiple bitrates in ABR algorithms. Each bitrate video is divided into multiple small video chunks. ABR algorithms will dynamically choose a bitrate for each video chunk. The bitrate selection is based on various observations such as the network throughput and playback buffer occupancy. The goal is to maximize user Quality-of-Experience (QoE) by adapting the video bitrate based on the underlying network conditions. However, limited network bandwidth resource may lead to poor user QoE of UHD video delivery even state-of-the-art ABR algorithms are adopted.

With the magic of Graphics Processing Units (GPUs) and deep learning techniques, many approaches try to use deep neural networks (DNNs) to improve the performance of video delivery and user QoE. For instance, a compression framework based on convolutional neural networks (CNNs) is proposed to achieve high-quality image compression at low loss rates [7]. Deep reinforcement learning (RL) models are integrated with ABR algorithms to optimize user QoE. However, these works are mainly focused on improving the performance of existing ABR systems. The

challenges brought by UHD videos are not considered.

1.1.2 Real-time Video Streaming Applications

In recent years, live video streaming has become tremendously popular and will account for 17% of the Internet video traffic by 2022 [8]. Live video streaming services allow users to stream live videos over the Internet and to interact with their viewers. There are many live video streaming applications, such as remote education, live sports event, game streaming, and journalism. For these applications, viewers will typically tolerate tens of seconds of delay in video streaming. However, for some live video streaming applications, such as remote control and video conferencing, participants require very low latency for interactivity (less than one second [9]). That is why these applications are called *real-time* video streaming applications. It is important but challenging to stream high-quality videos while having very low latency.

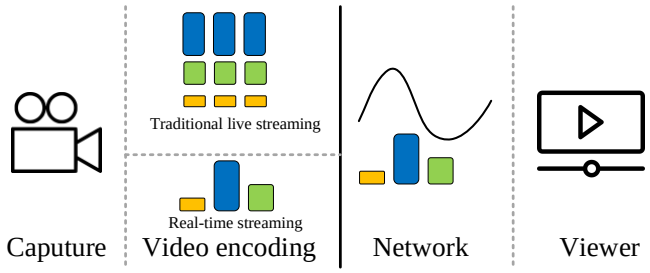


Figure 1.2: The live video streaming system.

In traditional live streaming applications [10, 11], the captured raw video frames are encoded into a video according to the setting of several key parameters (e.g., bitrate). Adaptive bitrate video encoding algorithms (ABR) are widely used in live video streaming systems[5, 6, 12]. As shown in Figure 3.1, in ABR based live video streaming, a video is encoded into multiple tracks. Each track is encoded with a different bitrate or quality level. Each track video is then divided into multiple small video chunks, and each chunk contains a few seconds of the video content. These video chunks are uploaded to the content distributed network (CDN) for a large

group of receivers. Traditional live streaming applications can usually tolerate longer latency, as compared with real-time streaming applications. Thus, the encoding time of multiple video tracks is usually ignored.

In recent years, many ABR algorithms adopt deep learning techniques to make the best bitrate selection. Applying these deep-learning based ABR algorithms has greatly improved the performance of live video streaming systems. Several network protocols based on ABR are designed for live video streaming, such as DASH, HLS, and RMTP [13, 14, 15]. These network protocols can dynamically determine which bitrate to transmit for each chunk position in the video. The bitrate selection is made based on various observations, such as network throughput and user playback buffer occupancy. The key idea of these protocols is to maximize the user QoE by adapting the video bitrate according to different network conditions.

On the contrary, real-time video streaming applications [16] usually use point-to-point communications or there will be only a small group of participants (CDN based video streaming structure is unsuitable for real-time video streaming). In this case, generating multiple video tracks is not necessary and will introduce additional latency. Instead, only one video stream will be generated. In addition, the video encoding time cannot be ignored in real-time streaming applications, because a very low latency is required for real-time interaction between the broadcaster and the viewers. Each step that introduces latency should be considered. The Real-time Transport Protocol (RTP) and congestion control algorithms are widely used in these applications [17, 18]. For instance, WebRTC is one of the most popular protocols to achieve real-time video communications, which uses the google congestion control (GCC) algorithm [19] to adapt the bitrate of the video stream to the network condition (e.g., change the frame resolution). However, the main limitation of these approaches is that the quality of the video streaming will be sacrificed to satisfy the real-time requirement [10]. Therefore, meeting the real-time requirement without sacrificing the video quality,

or even further improving the user QoE is an important research issue and has not been addressed for real-time video streaming applications. In addition, since existing ABR algorithms are designed for the on-demand streaming of pre-recorded video applications, directly applying these ABR algorithms to real-time video streaming applications cannot address the above research issue well and also introduces many challenges.

1.1.3 Mobile Augmented Reality Applications

Mobile augmented reality (AR) applications attract more and more attention in recent years. Companies are developing AR platforms and devices such as Google ARCore and Microsoft HoloLens to encourage developers to create mobile AR applications. We can see an increasing market size of AR applications. However, one of the most important challenges is the limited resources on mobile devices, such as CPU, GPU, memory, battery, and storage.

Mobile AR applications need a large amount of computation resources to support the scene analysis and interactions with users. For instance, in a smart shopping application [20], the mobile device needs to recognize the product, including its shape, color, and brand. Another example is the virtual tourism application like HoloTour [21]. HoloLens will recognize the gesture or voice commands so that a user can interact with the virtual objects. To recognize the objects and human actions, advanced computer vision algorithms will be deployed. Running these algorithms requires high complexity computation, and a large amount of computation resources are required.

Existing mobile AR systems can be mainly classified into two categories. The first one is the cloud/edge-based mobile AR system [22, 23, 24, 25]. They choose to offload recognition tasks to powerful servers. As shown in Figure 1.3, the mobile device is used to capture the scenes and visualize the results. The captured data will be uploaded to servers including both the edge and cloud server. These servers are used

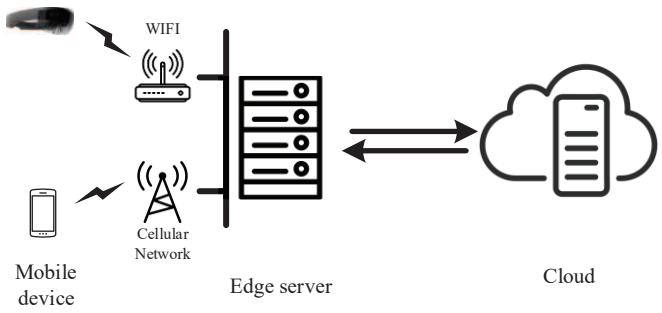


Figure 1.3: The cloud/edge-based mobile AR system.

to perform image analysis using computer vision algorithms. The main advantage of cloud/edge-based mobile AR systems is that we can apply advanced recognition algorithms with high precision on the servers. However, current cloud/edge-based mobile AR systems cannot achieve fast response which is a vital feature of mobile AR applications. The reason is that offloading will cause high network latency.

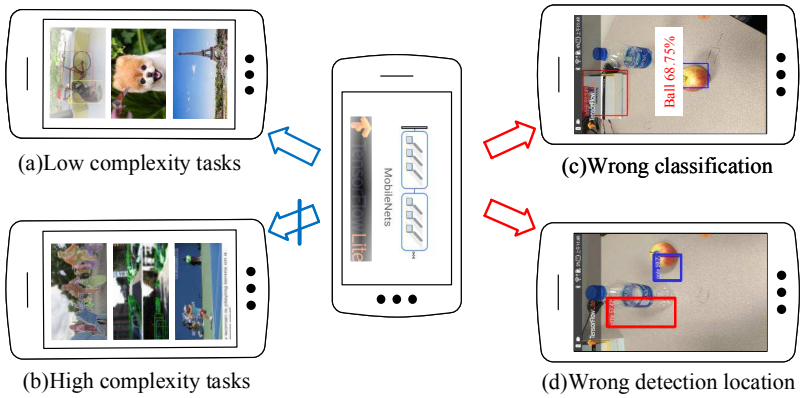


Figure 1.4: The mobile AR applications challenges.

The second one is the on-device mobile AR system [26, 27, 28]. Before we are able to implement the deep learning based algorithms, the recognition tasks are achieved with low complexity methods such as image feature extraction, classification, and object tracking. The computation complexity of these methods is lower enough to be supported by mobile devices. However, with the development of the hardware (GPU on mobile devices) and software (Google MobileNets and Tensorflow Lite), the simple deep learning based algorithms can be directly deployed on mobile devices for some recognition tasks such as the classification and object detection. The advantage

of the on-device architecture is that we can do the recognition tasks on the device. There is no network latency involved. However, there are several challenges to apply the on-device models. As shown in Figure 1.4. The first one is that not all the deep learning models can be implemented on mobile devices for mobile AR applications. The low complexity tasks such as image recognition, object detection, and landmark recognition can be implemented on mobile devices. The high complexity algorithms such as object segmentation, 3D object detection, and image semantic analysis can only be supported by the edge server and cloud. Secondly, the recognition performance is not robust. For instance, the on-device models produce wrong classification and detection results very frequently. We can only get satisfied recognition results in some specific scenes and applications.

1.2 Introduction to the Edge Assisted System

In an edge-based system, as shown in Figure 1.5, tasks are executed on the mobile device first. If the offloading decision algorithm finds that offloading can bring higher user QoE, the tasks are offloaded to the edge server. The server then executes the offloaded tasks and sends back the execution results to the mobile device. The main advantage of this edge-based system is that it combines the low latency of executing tasks on mobile devices and the high performance of executing tasks on the edge server.

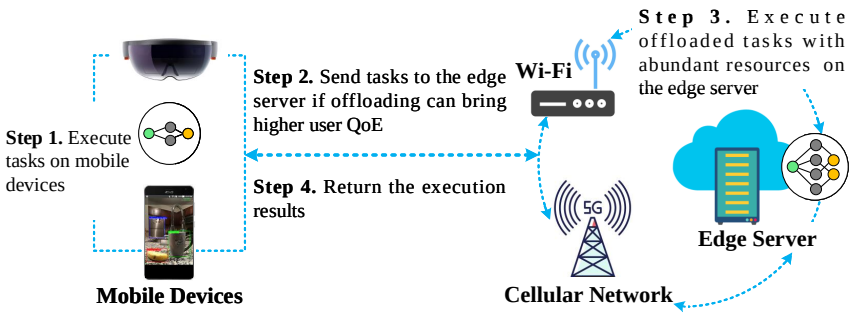


Figure 1.5: The system of edge-based MAR applications.

Offloading strategies are widely used in existing edge assisted systems [29, 30, 31]. Existing offloading decision algorithms try to balance the quality and latency of executing the tasks on the mobile device or on the edge server. However, a disadvantage is that if a task is offloaded to the edge server, task offloading and result feeding back introduce additional network latency. Longer latency not only degrades user QoE, but also decreases the performance of the offloaded tasks because the location and the environment of the user may have changed when the mobile device receives the results.

1.3 Ultra-dense Mobile Networks

In a wireless communication network, a massive number of mobile devices produce vast amounts of data every day. With the growth of Internet of Things (IoT) and Virtual Reality (VR)/Augmented Reality (AR) applications, the wireless network data will keep increasing exponentially. This brings both the opportunity and challenge to wireless networks. The opportunity offered by wireless big data is that we can improve the network performance by discovering useful information from the network big data. With the wireless big data analysis, we can predict traffic loads of wireless networks [32], perform the network planning and optimize the network resource allocations [33]. On the other hand, the growth of mobile devices and applications will keep enlarging the demand for wireless network capacity [1]. To carry the ever-increasing mobile data, radio base stations (BSs) are being densely deployed to provide high network capacity [34, 35]. Although the ultra-dense network promises high network capacity, it significantly increases the complexity of networking functions such as the traffic load balancing and BS sleep control because of the large number of BSs [36, 37]. The rapid growth in complexity stifles existing network functions for ultra-dense mobile networks. For instance, the computational complexity of the suboptimal traffic load balancing function can be estimated to be $O(M^a N^b)$ for some nonnegative integer a and b, where M and N are the numbers of users and BSs,

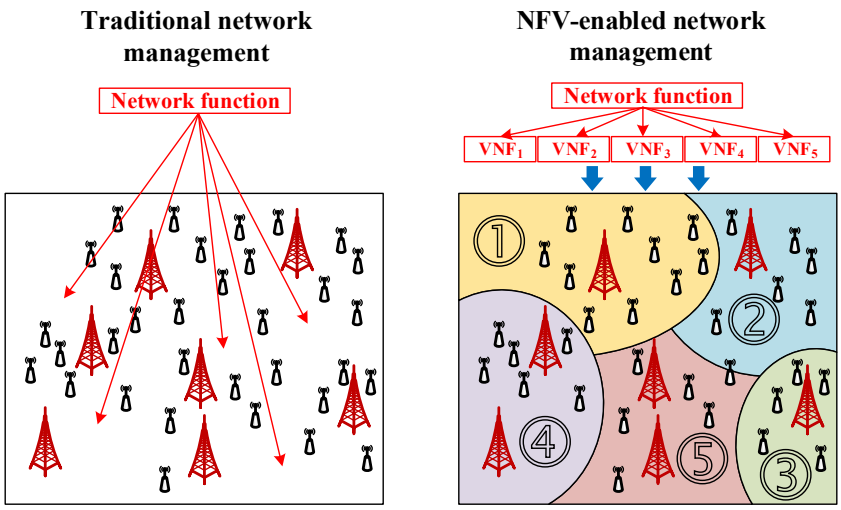


Figure 1.6: The VNF placement.

respectively [38]. When BSs are densely deployed, these network functions can incur long computing time and high communication overhead [36, 39, 40].

Network function virtualization (NFV) is an emerging networking technology for managing complex communication networks. With NFV, network operators can dynamically place network functions to facilitate efficient network management [41, 42, 43]. In an ultra-dense mobile network, a network function cannot efficiently manage the network because of the high computational complexity. They propose to use a divide-and-conquer method that partitions the network into multiple sub-networks and deploys replicas of the network function to manage each sub-network independently. The research challenge is how to partition the network into how many sub-networks.

Figure 1.6 compares the traditional and NFV-enabled network management. In the traditional network management, a network function manages the whole network, which will lead to a long computation delay in searching for the optimal network management solution. With the proposed NFV-enabled network management, the whole network can be partitioned into multiple sub-RANs, and each sub-RAN is managed by a replica of the network function. In this way, the computational complexity of

the network function can be significantly reduced, and thus the computation delay is shortened. However, partitioning RAN is challenging for a few reasons. First, the relationships among BSs cannot be accurately quantified [38]. Hence, it is non-trivial to define the distance function that quantifies the relationships among BSs for the partitioning. Second, owing to the dynamic mobile traffic, it is difficult to learn the optimal network partitioning solution at different time intervals.

1.4 Overview of the Proposed Research

Figure 1.7 shows the overview of the proposed research. We focus on two types of resource-hungry applications: video streaming and mobile AR. For video streaming applications, a dynamic video encoding configuration protocol is proposed for real-time video streaming applications, and a cloud computing based deep compression framework is proposed for UHD video streaming applications. For mobile AR applications, a smart task allocation decision algorithm, a frame-level offloading decision algorithm, and a fast model updating algorithm are proposed to improve the performance of MAR applications. Furthermore, a network partitioning algorithm and an online network optimization framework are proposed to tackle the issues of ultra-dense mobile networks. These proposed research targets on some unique and practical issues regarding user QoE in resource-hungry applications.

For video delivery applications, we propose two research works. First, we propose DAVE, a dynamic video encoding protocol for real-time video streaming applications. DAVE uses a reinforcement learning based model to study the optimal video encoding configurations under different network conditions in order to achieve the optimal performance of real-time video streaming systems and user QoE. DAVE also uses super resolution algorithms to improve the video perceptual quality without increasing the video transmission data size and introducing additional latency.

Then, we propose Pearl, a cloud computing based deep compression framework that applies deep learning techniques to UHD video content compression to maximize UHD

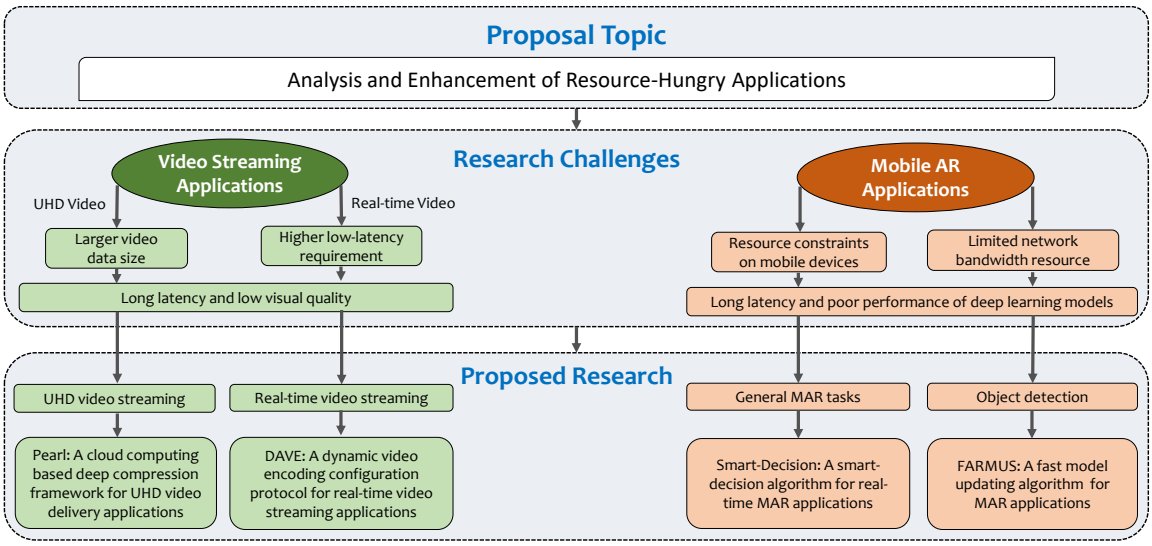


Figure 1.7: The overview of the proposed research.

video delivery performance and user QoE. In pearl, an optimal compact representation of the original UHD video is learned with two deep convolutional neural networks (DCNNs): super resolution CNN (SR-CNN) and colorization CNN (CL-CNN). SR-CNN is used to reconstruct a high resolution video from a low resolution video while CL-CNN is adopted to preserve the color information of the video. With Pearl, the data size of UHD videos can be significantly reduced.

For MAR applications, we propose three designs. First, to improve the effectiveness of on-device object detection, we propose an improved frame-level offloading decision algorithm (FLOD) that can reduce the occurrence of object detection-failure and improve the user QoE.

Then, to achieve fast model updating and satisfy the diverse requirements of different MAR applications, we propose a new online model updating scheme, FARMUS, for MAR applications. We formulate a model updating decision problem to determine the proper moment for model updating and take user preference and user movement pattern into consideration.

Next, we proposed a smart-decision framework which combines the advantages

of the on-device mobile AR system and the edge-based system to achieve real-time recognition tasks. The framework will decide where to execute the recognition tasks. To overcome the dynamic changes of network condition and the limitations of the on-device deep learning models, we design a cache and matching algorithm on mobile devices. Compared with other related works, we focus on enhancing the performance of mobile AR applications.

For ultra-dense mobile networks, we propose two specific designs. First, a data-driven network partitioning algorithm is proposed to address the high complexity of networking mechanisms. We mined a mobile traffic data set containing billions of mobile traffic records to analyze the relationships among BSs. Three networking features are extracted from the mobile traffic data set. These features are the geo-distance between BSs, the similarity of traffic trends in BSs, and the common active users shared in BSs during a certain period of time. Based on these networking features, we design a network partitioning framework.

Then, we propose a data-driven network optimization (DINO) framework which integrates data analysis and machine learning methods with network optimization algorithms to enable realtime and efficient ultra-dense network optimization. The proposed framework is composed of an offline machine learning module and an online BS clustering and a network optimization module. The offline module mines the mobile traffic dataset and performs BS clustering analysis using the hierarchical clustering analysis (HCA) method [44]. The analysis results are utilized to train an artificial neural network (ANN) to predict the optimal number of BS groups at different time intervals. Based on the prediction, the online module adopts the K-medoids clustering algorithm to partition BSs into groups and then optimize the performance of each group of BSs independently.

1.5 Dissertation Organization

The rest of the research is organized as follows. In Chapter 2, related work on the proposed research is introduced. In Chapter 3, DAVE, a dynamic video encoding configuration protocol is proposed. Next, Pearl, a cloud computing based deep compression framework for UHD videos is presented in Chapter 4. Then, an improved frame-level offloading decision algorithm, a fast online model updating scheme, and a smart-decision algorithm are presented in Chapter 5. After that, an offline network partitioning algorithm and an online network partition optimization model are presented in Chapter 6 to address the issues in ultra-dense mobile networks. Following that, the publications and future work are listed in Chapter 7.

CHAPTER 2: RELATED WORK

In this chapter, existing research on video streaming applications and mobile AR applications is discussed followed by the existing work on ultra-dense mobile networks. Existing research approaches to enhance the performance of resource-hungry applications and their shortcomings are also described in this chapter.

2.1 Existing Research on Video Streaming Applications

2.1.1 Video Encoding and Decoding

Video is defined as a continuous series of frames. Frame resolution, frames per second (FPS), bitrate, video compression rate, and video encoding speed are the most important factors of video encoding. To improve the video transmission QoE, ABR algorithms are widely adopted to handle the dynamic change of the network in real world. Traditional ABR algorithms usually fix all other encoding parameters, except the video compression rate, to encode a video into multiple bitrate tracks [3, 45]. If the bitrate of these tracks still cannot satisfy the requirements, the resolution of the video will be downsampled to reduce the bitrate of the video tracks. However, by configuring other factors, we can achieve higher video quality with the same bitrate or achieve the same video quality with a lower bitrate. Video Multimethod Assessment Fusion (VMAF) is a widely used video perceptual quality assessment algorithm. A VMAF score is between 0 and 100: 0-20 is considered as unacceptable, 20-40 as poor, 40-60 as fair, 60-80 as good, and 80-100 as excellent [46].

For each video frame, the most commonly used color mode is RGB, which means each frame can be divided into three channels (R, G, and B). However, a video is not composed of original RGB frames. To save data storage space, video frames need to

be compressed and encoded into a video. To achieve better compression performance, the color mode of video frames is transformed from RGB to YUV in most of the video codecs, such as H.26x, and VPx [47, 48]. In all of the exiting super resolution based video delivery systems, video frames are extracted (decoding) from the video with RGB color mode. After applying the super resolution model, the generated RGB video frames need to be encoded into video again. Different from these systems, we proposed using YUV channel-based super resolution model. In Pearl, the results generated by the super resolution models are channels in YUV color mode. As a result, Pearl can reduce video processing latency.

2.1.2 Super Resolution

The super resolution algorithm aims to upscale and improve the perceptual quality within an image [6, 49]. A low-resolution image is taken as an input to the super resolution algorithm, and a higher resolution image with higher perceptual quality is the output. Recent research on super-resolution has achieved great progress with the development of deep convolutional neural networks. The speed of generating a 1080p resolution frame with the super resolution model proposed in [50] is 34 FPS (larger than 24 FPS), which means that we can apply the super resolution model on the low-resolution videos to improve the video quality in the real-time video streaming system. With the help of model compression techniques, super resolution models can be deployed on mobile devices.

There are several state-of-the-art super-resolution algorithms, such as EDSR and ESRGAN [51, 52]. The main obstacle to adopt these algorithms is the time complexity. For example, the inference speed of EDSR for 1020x768 frames is 2.08 frames per second, which makes it impossible to achieve real-time video processing. To meet the real-time constraint for upscaling the high-resolution frames, the fastest super resolution model FSRCNN-S [50] is adopted in this paper. For a super resolution model, we can choose the scales of up-scaling. {x2, x3, x4, x6} are widely used in

super resolution algorithms, where xk means upscaling both the width and height k times.

2.1.3 Colorization

There have been many studies about video colorization [53, 54]. Colorization can be used in many popular applications such as colorizing the old black and white photos, remastering classic black and white film [55, 56]. There are two types of colorization algorithms: user-guided colorization and data-driven automatic colorization. The limitation of the automatic colorization model is that we need to train individual models for different videos. For the user-guided colorization model, the main challenge is the generated images are closer to the referenced frame as compared with the original frame.

2.1.4 UHD Video Streaming Systems

There have been many works that apply DNN models on image compression [57, 58, 7]. The results show that DNN-based image compression outperforms traditional image compression algorithms. However, there are only a few studies about applying the DNN on video compression [59]. It is very challenging for deep learning models to achieve the same level of inter-frame compression performance compared with non-DNN based video encoding algorithms. Recently, Super resolution has been widely used for video compression [6]. Instead of directly compressing the video content, the image resolution of the video is downscale to reduce the video data size in these systems. As a result, lower network transmission latency and less network bandwidth resource usage can improve user QoE. However, all of these systems are focusing on 2k videos. The challenges brought by UHD videos are not addressed. To the best of our knowledge, Pearl is the first framework that applies deep learning driven compression on UHD videos.

2.1.5 Real-time Video Streaming Systems

Adaptive bitrate algorithms are designed for handling the dynamic network throughput in the real world. Rate-based and buffer-based approaches are the earliest ABR algorithms. Rate-based ABR algorithms [3, 45] make the bitrate selection based on the predicted network bandwidth. Buffer-based ABR methods [60, 12] select the bitrate according to the playback buffer occupancy of the client player. Recently proposed ABR algorithms combine the rate-based and buffer-based methods. The state-of-the-art ABR algorithm [5] uses deep learning models to further improve the performance of ABR systems. However, these ABR algorithms ignore both the impact of the video encoding configuration and the video encoding time, which are necessary for live video streaming systems. Moreover, the ABR algorithms usually use the information of future video chunks to optimize the system performance, which is not practical in real-time video streaming applications. Hence, we need to design a new protocol for optimizing real-time streaming systems.

Existing live video streaming systems apply several specifically designed network protocols to improve the system performance [13, 14, 15, 16]. These network protocols mainly focus on optimizing the video packetization and network transmission (e.g., lower packet loss rate and better video bitrate selection). Compared with these works, DAVE adopts an end-to-end optimization strategy to further improve the performance of real-time video streaming applications.

There are some works that study the relationship between the video encoding parameters (e.g., FPS) and video quality [61, 62]. However, they only focus on the video perceptual quality. The video encoding time and video data size are not considered. We study the characteristics of video encoding configurations with a rich set of videos. The tradeoffs among the video data size, video perceptual quality, and video encoding time are well learned.

Super resolution has been used in a variety of computer vision applications, in-

cluding video enhancement, medical diagnosis, and surveillance [51, 52, 63]. The performance of super resolution algorithms has been greatly improved with the help of deep convolutional neural networks. However, super resolution models are usually very time and GPU memory consuming. To overcome this challenge, many light super resolution models are proposed [50, 6].

2.2 Existing Research on Mobile Augmented Reality Applications

2.2.1 Image Features and Object Tracking

Image Features are numerical descriptors that individually or collectively form unique signatures to describe the image. Typically, image recognition techniques rely on discovering multiple interesting points in the image and extracting descriptors for these points. These interesting points collectively form a signature for the image. This technique is used by many feature detection and extraction techniques such as SIFT [64], SURF [65] and ORB [66]. **Object tracking** is widely used in Mobile AR/VR systems [22, 23, 25] before the invention of deep learning algorithms. The advantage of adapting object tracking is that once you recognize the objects, you don't need to recognize them again if you can keep tracking the objects based on their unique features. In this case, a lot of redundant computations are void and using the tracking algorithm is fast enough to achieve real-time processing.

2.2.2 Recognition Tasks

The **recognition** is vital for mobile AR applications. Image recognition is used for identifying the objects in the image and analyzing the semantic information(like human actions) in the image. It can help the device to understand what you see and what you want to do, which is the first step to allow the interaction with the environment. Recognition can be divided into several small tasks, such as object detection, object segmentation, semantic analysis, and so on. State-of-the-art recognition algorithms[67, 68, 69] can achieve high-quality results, and they all use deep convolu-

tional neural networks (CNN). Unfortunately, these deep and complicated networks have significant computation and memory needs. In this case, deploying them on mobile devices is very challenging. For instance, SSD300 and Faster-RCNN300 need 34.9 and 64.3 billions of Multi-Add calculations; 33.1 and 138.5 millions of parameters need to be calculated and maintained, respectively.

In the computer vision field, there are some works that try to build light deep neural networks to balance the latency and accuracy of the detection and recognition models. For instance, Howard *et al.* [70] presents an efficient model-MobileNets for mobile and embedded vision applications. Although it reduces about 80% computation complexity of current object detection models, it still needs to process millions of parameters, which cause billions of Muti-Adds calculations. Apte *et al.* [71] deploys the tiny-YOLO model on iPhone 7 to achieve the real-time object detection. It can achieve 8-11 FPS, but the accuracy is low. With the invention of Tensorflow Lite, some recognition algorithms can be deployed on mobile devices with high accuracy. The speed of running the object detection task on android mobiles phones is about 5-6 FPS.

2.2.3 Mobile AR Systems

There are a lot of works related to mobile AR systems. Chen *et al.* [23] proposed a system called Glimpse which combines tracking and caching to achieve object detection on mobile devices. However, it can only detect the pre-trained objects such as road signs and human faces. Drolia *et al.* [24] developed an edge caching solution for image recognition. It will predict the objects which a user might meet in a short time and fetch the classification model from the edge server. However, this solution can not get the locations of the objects, which is an essential function for mobile AR applications. Zhang *et al.* [25] proposed a cloud-based framework, CloudAR, to achieve real-time image detection based on object tracking. They segment and track the objects on the client side and do the detection on the server. Ran *et al.* [29]

proposed DeepDecision to determine an optimal offloading strategy for AR tasks.

However, these works mainly focus on making the decision of when to offload the tasks to the edge server, they do not take the performance of the on-device models into consideration. We focus on enhancing the performance of mobile AR applications. A cache and matching algorithm will be used when the performance of on-device deep learning models is poor.

2.2.4 Offloading

Offloading in edge/cloud computing systems is widely used to address the issues caused by the limited resources on mobile devices and is adopted in most MAR systems [22, 23, 25, 72]. All of these systems choose to offload object detection tasks to the edge/cloud server. At the same time, all of these systems adopt tracking algorithms to reduce the frequency of offloading. However, these systems are proposed before the development of on-device object detection models. The advantages of the on-device object detection model are not utilized.

The optimal offloading decision problem has been well studied [73, 74, 75]. Offloading decision optimization has also been considered in MAR applications. DeepDecision is proposed in [29] to determine an optimal offloading strategy for AR tasks. The model accuracy, video quality, battery constraints, network data usage, and network conditions are considered to determine whether to execute the recognition tasks on the mobile device or offload to the cloud server. Another offloading optimization strategy is proposed in [30] that considers the tradeoffs between the frame resolution and the object detection model accuracy. However, they all focus on optimizing the decision to offload or not. In addition, mAP of detection models is used as the accuracy of detection for every frame, which may lead to inaccurate offloading decisions. We aim to reduce the occurrences of offloading caused by object detection-failure and miss-detection.

2.2.5 Object Tracking and Feature Matching

Object tracking is widely used in MAR systems [22, 23, 25]. Once an object is detected, the object can be tracked based on the image feature matching algorithm. Image features are the numerical descriptors that individually form the unique signatures of an object. The advantage of object tracking is that once an object is detected, it is not necessary to detect it again if the object can be tracked based on its unique features. In this way, repeated computations are avoided. However, there are several limitations of object tracking algorithms. For instance, it is very challenging to track multiple objects at the same time, while DNN based object detection algorithms can detect dozens of objects within one processing. Moreover, object tracking can only track the same object, but cannot detect a new object which belongs to the same class of the tracked object. Additionally, in order to track an object, the object needs to be detected first. When on-device miss-detection occurs, the object detection task offloading is inevitable.

2.2.6 Fine-tuning and Incremental Learning.

Fine-tuning is widely used for updating a new model based on an existing base model [76]. The new model can have different configurations from the base model, such as the number of classes. To incrementally add new object classes into the model, a straightforward way is to fine-tune the model with training data from both old and new classes. The challenge of applying fine-tuning is that it will take several hours of background time to train the new model, which is unacceptable for MAR users. Incremental learning is proposed in [77] for training new models for MAR applications. Compared with fine-tuning, incremental learning is much faster. However, the new model trained with fine-tuning is more robust. Both methods can be used to update the object detection model on mobile devices for MAR applications. However, currently there is no existing work on how to properly adopt a model updating strategy

in an MAR system.

2.3 Existing Research on Ultra-dense Mobile Networks

2.3.1 Virtual Network Function Placement

Our research work relates to the virtual network function placement and big wireless data analysis. The existing research solutions on virtual network function placements on wireless radio access networks are mainly designed for cloud computing and resource management [78, 79, 80, 81, 82, 83, 84, 85]. In these solutions, the network function placement is usually formulated as an optimization problem, which can be solved by either disciplined optimization problem solvers or heuristic algorithms [42, 86, 87, 88, 89, 90, 91, 92]. For instance, Zhang *et al.* [86] proposed a routing algorithm for jointly placing NFVs. The network is defined as a graph, and then they built an optimization problem to minimize the operational cost. Luizelli *et al.* [87] formulated the network function placement and chaining problem and proposed an Integer Linear Programming (ILP) model to solve it. These works can be used to improve the performance of the NFV systems. However, these optimization methods cannot be applied in an ultra-dense radio access network because of the complexity to obtain the optimal solution.

2.3.2 Big Data Analysis for Wireless Network

Big data analysis and machine learning have been applied to manage and optimize wireless communications networks. Most of the research works on big-data-driven mobile networks focus on developing big mobile traffic data monitoring and analysis systems. For example, Liu *et al.* [93] proposed a Hadoop-based scalable network traffic monitoring and analysis system for monitoring, collecting, and storing the big mobile traffic data.

Another research direction in the wireless big data analysis is to discover the patterns of the network usage and user behavior through analyzing big mobile traffic

data. For instance, by mining a big mobile network traffic data set, Xu *et al.* [94] studied the correlation of the human mobility and mobile traffic data consumption for a better understanding of human behaviors. Su *et al.* [95] proposes a framework for delivering interest information to users by combining the social network analysis and mobile big data analysis.

2.3.3 Wireless Network Optimization Frameworks

A few research works have applied wireless data analysis to optimize various aspect of mobile networks. Wang *et al.* [32] proposed a cellular network traffic load prediction model based on big data analysis, in which they focused on using deep learning to predict traffic loads in order to utilize the network resources efficiently and effectively. However, they did not investigate how to use the prediction results to optimize the network performance. Zheng *et al.* [33] proposed a conceptual framework, which enables big data analysis for the mobile network optimization. They presented a conceptual framework by using big data and machine learning to optimize the network, and illustrated some use cases like network planning, resource allocation, and interference coordination. Mehraghdam *et al.* [80] proposed a system to collect the network traffic statistics, and to conduct application-layer analysis, web service provider analysis and user behavior analysis. These functions can be used to improve user QoE. All these works are position papers that describe conceptual frameworks of data-driven cellular network design. They only stated what functions can be done, but they did not provide any technical details and analysis on how to achieve these functions, what kind of data to be processed and what kind of algorithms to be adopted. To our best knowledge, wireless big data analysis has not been investigated to help place and manage the virtual network functions in ultra-dense radio access networks.

In our proposed framework, we partition the network into many sub-RANs. Although there are many existing works on wireless network partitioning, most of these

works focus on clustering mobile ad hoc and wireless sensor networks [96, 97, 98, 99, 100, 101]. These solutions are designed based on graph theory and node routing information and not appropriate for clustering cellular networks. Cellular wireless networks are deployed as infrastructure. A cellular network is a centralized structure. It consists of central entities known as base stations, and all BSs are connected to a core network. A base station provides service for a specific area of users. There are overlapping areas for nearby BSs. Controlling one BS also affects its nearby BSs with overlapped coverage areas. So, we cannot just simply model a cellular network as a graph. The existing solutions for cellular network partitioning rely on the geographic distances among BSs to cluster the BSs into different sub-networks [99, 100]. However, the relationship among the BSs cannot be accurately measured when different network functions are considered [38]. In this case, we propose a network partitioning solution based on wireless big data analysis.

CHAPTER 3: Proposed DAVE for Real-time Video Streaming

In this chapter, we propose a new real-time video streaming protocol, DAVE (Dynamic Adaptive Video Encoding for real-time video streaming applications). In DAVE, we propose a super resolution based video encoding configuration selection algorithm which does not use a fixed strategy to determine the encoding configurations as in existing real-time video streaming systems but uses a reinforcement learning based model to learn the optimal video encoding configuration that includes the configuration of both regular video encoding parameters and the up-scale of super resolution models. As a result, DAVE can enhance the performance of real-time video streaming systems based on user QoE metrics. To the best of our knowledge, this is the first work that incorporates super resolution and reinforcement learning in the protocol design for real-time video streaming systems.

3.1 Problem Description

In traditional live streaming applications [10, 11], the captured raw video frames are encoded into a video according to the setting of several key parameters (e.g., bitrate). These settings are defined as encoding configurations. Adaptive bitrate video encoding algorithms (ABR) are widely used in live video streaming systems[5, 6, 102, 12, 103]. As shown in Figure 3.1, in ABR based live video streaming, a video is encoded into multiple tracks. Each track is encoded with different configurations. Each track video is then divided into multiple small video chunks, and each chunk contains a few seconds of the video content. These video chunks are uploaded to the content distributed network (CDN) for a large group of receivers. Traditional live streaming applications can usually tolerate longer latency, as compared with real-

time streaming applications. As a result, the encoding of multiple tracks is usually done on the server with substantial resources, and the video encoding time is usually ignored.

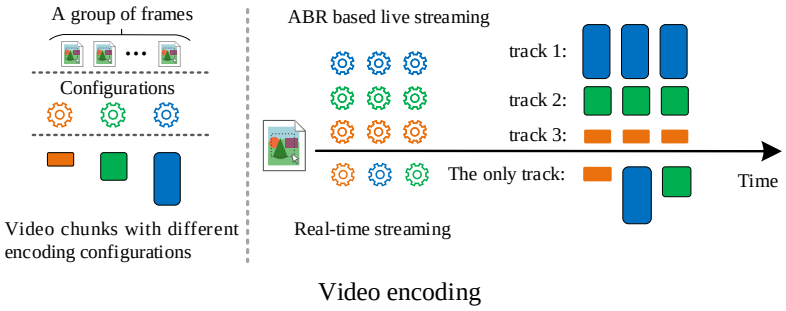


Figure 3.1: The live video streaming system.

In recent years, many ABR algorithms[102, 5] adopt deep learning techniques to make the best bitrate selection. Applying these deep-learning based ABR algorithms has greatly improved the performance of live video streaming systems. Several network protocols based on ABR are designed for live video streaming, such as DASH, HLS, and RMTP [13, 14, 15]. These network protocols can dynamically determine which bitrate to transmit for each chunk position in the video. The bitrate selection is made based on various observations, such as network throughput and user playback buffer occupancy. The key idea of these protocols is to maximize the user Quality of Experience (QoE) by adapting the video bitrate according to different network conditions.

Research Problem: On the contrary, real-time video streaming applications [16] usually use point-to-point communications or there will be only a small group of participants (CDN based video streaming structure is unsuitable for real-time video streaming). In this case, generating multiple video tracks is not necessary and will introduce additional latency. Instead, only one video stream will be generated. In addition, because the video has to be encoded in a real-time manner, the video encoding process is usually executed on user devices. Without the resources on the server or cloud, the video encoding time cannot be ignored because a very low la-

tency is required for real-time interaction between the broadcaster and viewers. The Real-time Transport Protocol (RTP) and congestion control algorithms are widely used in these applications [17, 18]. For instance, WebRTC is one of the most popular protocols to achieve real-time video communications, which uses the google congestion control (GCC) algorithm [19] to adapt the bitrate of the video stream to the network condition (e.g., change the frame resolution). However, the main limitation of these approaches is that the quality of the video streaming will be sacrificed to satisfy the real-time requirement [10]. Therefore, meeting the real-time requirement without sacrificing the video quality, or even further improving the user QoE is an important research issue and has not been addressed for real-time video streaming applications.

In addition, since existing ABR algorithms are designed for the on-demand streaming of pre-recorded video applications, directly applying these ABR algorithms to real-time video streaming applications cannot address the above research issue well and also introduces many challenges:

First, existing real-time video streaming systems usually reduce the video data size (for instance, reduce the frame resolution) to achieve the real-time requirement. Thus, the video perceptual quality will be sacrificed. With the help of deep learning techniques, we can solve this problem by transmitting low-resolution video frames and recovering them back to the high-resolution frames using super resolution algorithms [51, 52, 50]. The execution speed of a state-of-the-art super resolution model to recover 1080p videos is 34 frames per second (FPS) which can satisfy the real-time requirement of real-time video streaming applications [6]. In this case, the quality of the video streaming is improved. Another advantage of applying the super resolution algorithm is that the transmission data size and transmission latency will be reduced, since only low-resolution video frames are transmitted. However, the cost of applying the super resolution algorithm is that the user perception may be degraded.

The perception degradation level depends on the scale of the super resolution models. Larger scales will lead to higher perception degradation levels. Thus, there is a tradeoff between the video perceptual quality and the scale of the super resolution model. To integrate the super resolution algorithm with real-time video streaming applications, the challenge is to carefully configure the frame resolution setting.

Second, because the video has to be encoded in a realtime manner, how to determine the optimal video encoding configuration is a key issue. In traditional live streaming systems (ABR-based), videos are usually encoded with several fixed parameter settings, because of the efficiency and large-scale broadcasting. Thus, multiple video streams with different bitrates are generated, and the system is aware of all the video information, such as the data size of each video chunk. However, in real-time streaming applications, to achieve the low-latency requirement, only one video stream will be encoded in a real-time scenario. Thus, only a few seconds of the video can be accessed ahead at any moment, which means that less information can be utilized to make optimal streaming decisions. The state-of-the-art ABR algorithm (Pensieve [5]) uses the information of the whole pre-recorded video to achieve the optimal performance. However, this is not possible in real-time video streaming systems. Moreover, existing real-time streaming systems (e.g., WebRTC) use several fixed adaptation strategies to configure the video encoding. For instance, reduce the video FPS or frame resolution when the network throughput is lower than a certain level. However, these strategies lead to poor video quality (detailed in Section 3.2). Therefore, how to determine the optimal video encoding configuration without the future video information is a challenge.

Third, traditional live video streaming systems use bitrate to control the video quality: higher bitrate leads to higher video quality. However, according to our experiments (detailed in Section 4.4.2), the same video quality with a smaller bitrate can be achieved or the same bitrate can result in a higher video quality using different

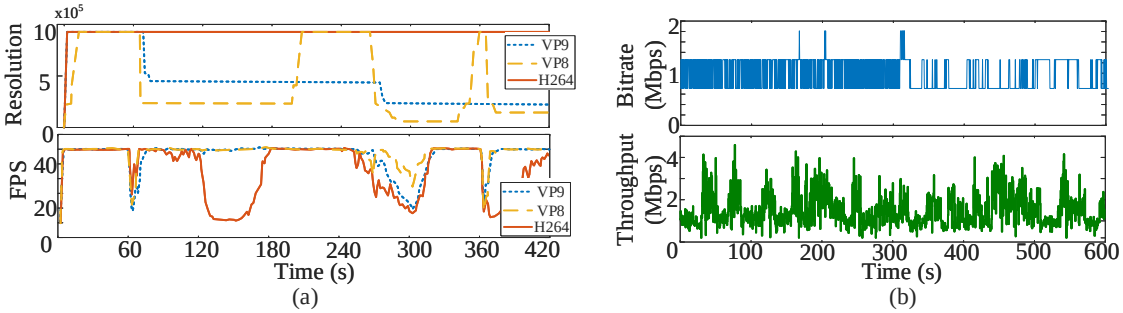


Figure 3.2: The performance of the WebRTC system (a) and ABR based live video streaming system (b).

video encoding configurations. This indicates that the potential of video encoding parameters is not well utilized. We can further improve the user QoE through dynamic video encoding configuration.

3.2 Motivation and Challenges

Applying ABR algorithms has greatly improved the performance of traditional live video streaming systems. However, current frameworks face several challenges when adopting ABR algorithms in real-time video streaming applications. To illustrate these challenges, we implement two experiments.

WebRTC is a real-time video streaming protocol that provides browsers and mobile applications with Real-Time Communication (RTC) capabilities. In WebRTC, the frame resolution and FPS are dynamically configured. However, the adaptation of the frame resolution and FPS follows fixed strategies. Following the setting in [10], we evaluate the performance of a WebRTC system. As shown in Figure 3.2(a), there is significant room for improvement in the adaptation scheme. For instance, the frame resolution of the video encoded with H.264 has not changed from 120 to 180 seconds. At the same time, the FPS is reduced under 5 FPS. For the video encoded with VP8 and VP9, a high frame rate is kept with a very low resolution from 300 to 360 seconds. The video quality is very poor under these configurations.

In the second experiment, we apply the simulator platform provided by the live

Table 3.1: The performance of the super resolution model

Resolution	Speed (FPS)	VMAF (after SR)	VMAF (before SR)
1080p	-	-	100
540p (x2)	8	94.93	88.04
360p (x3)	36	91.43	74.51
270p (x4)	42	85.56	59.02
180p (x6)	24	67.55	28.77

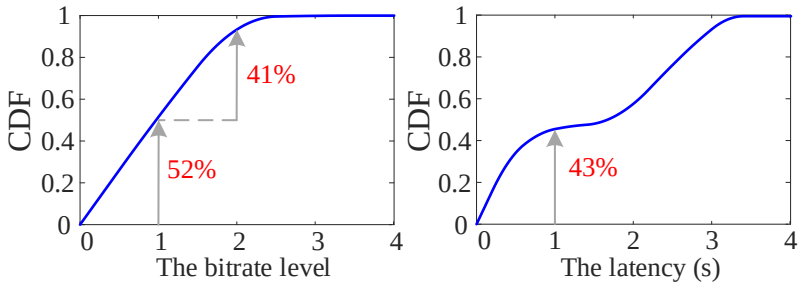


Figure 3.3: CDF of bitrate level and frame latency.

video streaming challenge at ACM Multimedia 2019 [104]. This simulator can be used to evaluate the performance of applying ABR algorithms in real-time video streaming systems. The ABR algorithm we selected is BBA [105] (one of the most robust ABR algorithms [106]). The result is shown in Figure 3.2(b). There are 4 video tracks: the bitrate of each video track is {500.0, 850.0, 1200.0, 1850.0}kbps. With the change of network throughput, different video chunks from different tracks are selected. As shown in Figure 3.3, 41% of the chunks are chosen from track 2, and 52% of the chunks come from track 1 (the lowest quality track). Video chunks from track 4 (the highest quality track) have never been selected. In addition, the end-to-end latency of 57% of the video frames are larger than 1-second (the latency requirement for the real-time video streaming applications [9]). These experimental results indicate that directly applying ABR algorithms in real-time video streaming systems will result in poor performance (latency and video quality). Thus, we cannot directly apply the existing ABR algorithms to optimize the performance of real-time video streaming applications.

According to the above two experiments, we can conclude that the video quality of the real-time streaming system which adopts ABR algorithms with fixed video encoding configurations is poor. These systems cannot achieve the optimal performance of real-time video streaming applications. Moreover, the end-to-end latency is a vital factor for real-time video streaming applications. However, the video encoding time is not taken into consideration in all the existing live video streaming systems. Different video encoding configurations will result in different encoding times. For instance, for a 1-second video chunk, the medium encoding time of all the configurations is 0.11 seconds. Since the average transmission time of a video chunk is 0.68 seconds, the video encoding time accounts for 14% of the end-to-end latency on average. In this case, we cannot ignore the influence of the video encoding time.

3.3 Characteristics of Video Encoding

In this section, we show a comprehensive study of the tradeoffs between the video encoding configuration and QoE metrics: video data size (bitrate), encoding time, and video perceptual quality. With the discovery from these experiments, we can design an optimal adaptive video encoding configuration selector.

3.3.1 QoE Metrics

The QoE metrics of real-time video streaming systems are mainly composed of two parts: network performance and video quality performance. The network performance is determined by the end-to-end latency. Unlike traditional live video streaming applications, the video encoding time has to be considered in real-time video streaming applications. So the network performance is defined as:

$$QoE_{network} = \sum_{n=1}^{N}(L_{encoding}(chunk_n) + L_{trans}(chunk_n)). \tag{3.1}$$

If we adopt the super resolution algorithm, the network performance is defined as:

$$QoE_{network} = \sum_{n=1}^{N}(L_{encoding}(chunk_n) + L_{trans}(chunk_n) + L_{SR}(chunk_n)). \quad (3.2)$$

The perceptual quality of a video can be calculated by the VMAF algorithm. However, two more factors need to be adopted to evaluate the quality of the video in real-time video streaming systems: rebuffering rate and smoothness. A rebuffering event will happen when the playback buffer of the client is empty and new content is still not delivered. This is caused by the cascading effects of bitrate selection (e.g., the network throughput decreases unexpectedly while a high bitrate video track is selected, which will lead to high transmission latency and cause rebuffering). Smoothness is defined as the adaptation frequency of the bitrate selection. Frequent bitrate changes will lead to poor QoE [5]. Thus, the video quality is defined as (3.3).

$$\begin{aligned} QoE_{video} = &\sum_{n=1}^{N} VMAF(chunk_n) - \mu_1 \sum_{n=1}^{N} T_n \\ &-\mu_2 \sum_{n=1}^{N-1} \left| VMAF(chunk_{n+1}) - VMAF(chunk_n) \right|, \end{aligned} \quad (3.3)$$

where T_n is the rebuffering time caused by downloading $chunk_n$. μ_1 and μ_2 are the penalty coefficients for rebuffering and smoothness, respectively. When we evaluate the characteristic of video encoding configurations, the video quality only refers to VMAF values. The reason is that rebuffering and smoothness are caused by ABR algorithms.

3.3.2 Video Encoding Settings

To investigate the impact of video encoding configurations on video quality, we download videos from 9 popular categories on Youtube (1: Beauty, 2: Comedy, 3: Cook, 4: Entertainment, 5: Game, 6: Music, 7: News, 8: Sports, 9: Technology). For each of these categories, we download three videos, each cut to 5 minutes in

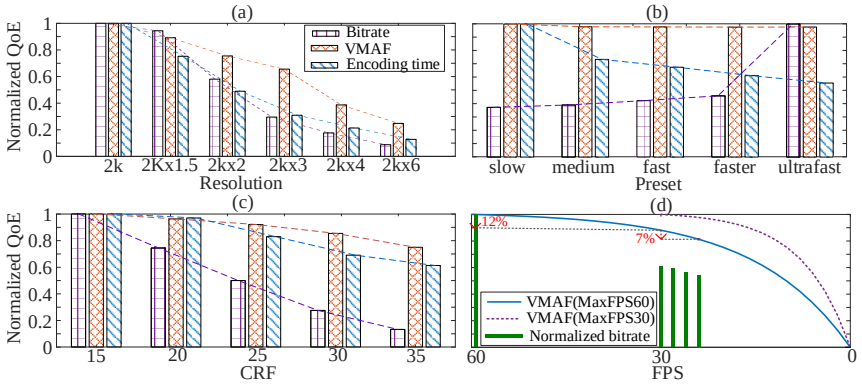


Figure 3.4: The impact of video encoding parameters on QoE metrics.

length. The video frames within 5 minutes are extracted (30FPS) to build a raw frame dataset for video encoding.

Table 3.2: The video encoding parameters

R	Frame resolution	$[2k, 2kx2, 2kx3, 2kx4, 2kx6]$
F	Frames per second	$[30, 28, 26, 24]$
C	Video compression rate	$[15, 20, 25, 30, 35]$
P	Video encoding speed	$[slow, med, fast, faster, ufast]$
G	Chunk interval	1

There will be millions of configurations if we traverse all the possible values for each video encoding factor, which is not practical. In this case, we select the most commonly used parameter values to build the configuration map, as shown in Table 3.2. To calculate the VMAF value of each configuration, a reference video with the highest quality is needed. In this paper, video encoded with {R:2k, F:30, C:15, P:slow, G:1}[1] is used as the reference video to measure the video quality of the other configurations.

3.3.3 QoE vs. Video Encoding Configuration

When we study the influence of each configuration parameter on the video quality, all other parameters are kept in the same setting. Thus, each parameter can be

[1]Encoded with FFMPEG: ffmpeg -r 30 -f image2 -s 1920x1080 -start_number 1 -i frame-folder/frame_%04d.jpg -vframes 9000 -vcodec libx264 -profile:v high -crf 15 -g 30 -preset slow -pix_fmt yuv420p result.mp4

validated independently. In Section 4.4, when we try to learn the optimal video encoding configurations, the entire configuration (a combination of parameters) will be evaluated instead of individual parameters.

Frame resolution: As shown in Figure 3.4(a), the video quality decreases with the downscale of the frame resolution. The VMAF values of video encoded with 2kx4 and 2kx6 are 38.61 and 24.78, respectively, which are classified as poor video quality. In the current live streaming systems, these poor quality options will be selected when the network throughput is low. However, we can avoid generating such low-quality videos by adopting different configurations. For instance, instead of reducing the frame resolution to an extremely low level, we can reduce the FPS to reduce the video data size and video encoding time while maintaining high video quality. The advantage of downscaling the frame resolution is that the encoding time and video bitrate significantly decrease with the downscale of the video resolution. The reason is that the video encoding time and data size not only depend on the Preset and CRF setting, but are also affected by the data that need to be computed. Low frame resolution videos have much smaller data that need to be encoded.

Preset: The changes of the Preset setting have an extremely small impact on the video perceptual quality. As presented in Figure 3.4(b), the worst VMAF value is 97, which indicates excellent video quality. The main tradeoff on Preset is between the video encoding time and video bitrate. Fast encoding options cannot fully utilize the power of the compression algorithms in the video encoder, which leads to a high bitrate. The fastest encoding option, $ultrafast$, almost doubles the bitrate compared to $faster$, with only a 6% improvement in the encoding time. Therefore, this option seems inefficient. However, there are still some special cases in which we can apply this option. For instance, when the client's buffer is empty and the network throughput is high.

CRF: CRF is used to control the compression rate of the video. A high CRF value

represents a high compression rate. A video encoded with a high compression rate will result in a small bitrate and low video quality. Compared with reducing the frame resolution, increasing the CRF value can reduce the bitrate to a similar level while keeping a much higher VMAF value. For instance, as shown in Figure 3.4(c), when the CRF is changed from 15 to 35, the bitrate is reduced by 86%, while the VMAF is reduced from 100 to 74, which is still classified as good quality. The disadvantage of CRF is that it requires more encoding time compared with reducing the frame resolution.

FPS: Compared with other parameters, we cannot directly measure the impact of FPS on video quality. This is because the VMAF value of a video is defined as the mean VMAF value of all the frames in the video. When the FPS is changed, the frame numbers cannot be matched between the encoded video and the reference video. In this case, we adopt a state-of-the-art algorithm [107] to evaluate the influence of changing FPS on VMAF. The algorithm is defined as:

$$Q(f, f_{max}) = \frac{1 - e^{-\beta_1 * (\frac{f}{f_{max}})}}{1 - e^{-\beta_1}} \frac{1 - e^{-\beta_2 * (\frac{f}{f_{max}})}}{1 - e^{-\beta_2}}, \tag{3.4}$$

where β_1 and β_2 are set to 4 and 5.5, respectively. The most commonly used maximum FPS is 30 and 60. We plot both cases in Figure 3.4(d). The VMAF is reduced by 12% and the bitrate is reduced by 40% when the FPS decreases from 60 to 30. The video encoding time is reduced by 9%. According to these observations, it is highly efficient to reduce the FPS instead of reducing the resolution, if the current FPS in the real-time streaming system is large than 30. Since the VMAF is lower than 60 (good) if the FPS is lower than 20. We should try to avoid encoding the video with an FPS value less than 20. When the maximum FPS is 30, the VMAF will be reduced by only 3% when the FPS decreases from 30 to 24. At the same time, 9% of bitrate and 16% of encoding time are saved. Therefore, reducing the FPS can contribute to improving

the real-time streaming system performance. However, we should be careful of the boundary value of FPS, because the video player will autofill black frames if there are not enough frames, which is worse than having enough frames with lower quality.

Summary: Through the tradeoff study, we can obtain guidelines to configure the video encoding parameters. However, it is still challenging to find the optimal video encoding configuration because of the dynamically changing network and system status. Existing systems (e.g., WebRTC) use a fixed strategy to configure the video encoding. However, there are still large rooms for optimizing the system as analyzed in Section. 3.2

3.4 Proposed Design and Implementation of DAVE

In this section, we detail the design and implementation of DAVE, a new protocol that applies reinforcement learning to optimize the video encoding configuration for real-time video streaming applications.

3.4.1 System Overview

To optimize the performance of real-time video streaming systems, a new optimized protocol is proposed, namely DAVE (Dynamic Adaptive Video Encoding). *In DAVE, the optimal video encoding configuration will be dynamically adapted to configure the video encoding.* As a result, DAVE can be more adaptive to the fluctuating network conditions and achieve better performance in real-time video streaming applications.

Sender: As illustrated in Fig. 6.2, the original video frames will be encoded with video codecs (e.g., H.264) to compress the video data size before packetization. The captured frames are encoded into a video stream according to the configuration settings. Then, the encoded video stream is packetized with network protocols (e.g., RTP). Once the encoded video streaming has been sent to the receiver, the user QoE and network statistics are collected and filtered. These information will be sent back to the optimizer at the sender. The optimizer will adopt the reinforcement learning

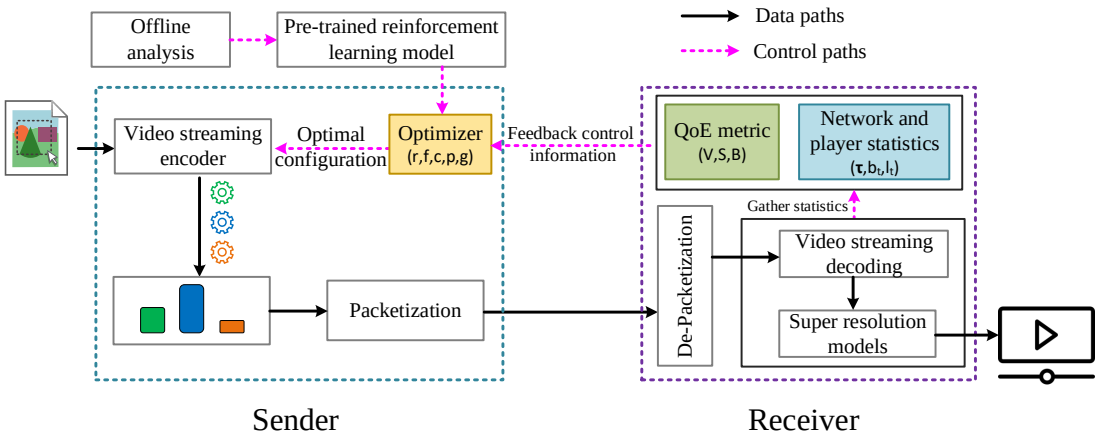


Figure 3.5: The proposed system framework.

model to predict the optimal video encoding configuration for the next time interval. We put the optimizer on the sender because video encoding is executed on the sender instead of the server or cloud, and the predicted optimal configurations can be instantly applied.

Receiver: After receiving the video streaming, the video frames are decoded. Then, if frames are down-scaled at the sender, the super resolution model is applied to upscale frames to the original size. The network statistics are collected after the frame is prepared to be rendered in the video player. The network statistics include the transmission time and throughput of each video chunk, the delay and the current buffer of the player. Additionally, the video player at the receiver side will calculate the video QoE metrics: rebuffering time (B), smoothness (S), and video perceptual quality (V). All of these measurements are sent back to the optimizer at the sender side which will predict the optimal video encoding configuration.

3.4.2 QoE Metrics

The QoE metrics of real-time video streaming systems are mainly composed of two parts: end-to-end latency and video quality. Unlike traditional live video streaming applications, the video encoding time has to be considered in real-time video

streaming applications. So the end-to-end latency is defined as:

$$Latency = \sum_{n=1}^{N}(L_{encoding}(chunk_n) + L_{trans}(chunk_n)). \tag{3.5}$$

If we adopt the super resolution algorithm, the end-to-end latency is defined as:

$$Latency = \sum_{n=1}^{N}(L_{encoding}(chunk_n) + L_{trans}(chunk_n) + L_{SR}(chunk_n)). \tag{3.6}$$

The perceptual quality of a video can be calculated by the VMAF algorithm. However, two more factors need to be adopted to evaluate the quality of the video in real-time video streaming systems: rebuffering time and smoothness. A rebuffering event will happen when the playback buffer of the client is empty and new content is still not delivered. This is caused by the cascading effects of bitrate selection (e.g., the network throughput decreases unexpectedly while a high bitrate video track is selected, which will lead to high transmission latency and cause rebuffering). Smoothness is defined as the adaptation frequency and adaptation amplitude of the bitrate selection. Frequent bitrate changes lead to poor user experience [5]. Thus, the video QoE metrics are defined as:

$$V = \sum_{n=1}^{N} VMAF(chunk_n), \tag{3.7}$$

$$B = \sum_{n=1}^{N} T_n, \tag{3.8}$$

$$S = \sum_{n=1}^{N-1} \left|VMAF(chunk_{n+1}) - VMAF(chunk_n)\right|, \tag{3.9}$$

where T_n is the rebuffering time caused by downloading $chunk_n$.

3.4.3 Reinforcement Learning Model

In this paper, we consider a learning-based approach to generate the optimal video encoding configuration. Unlike approaches that use fixed rules in the form of fine-

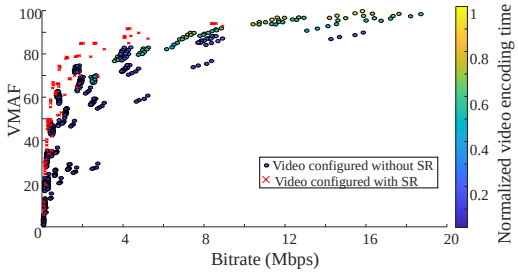


Figure 3.6: The quality of videos generated by different configurations.

tuned heuristics, we attempt to learn an optimal video encoding policy based on reinforcement learning (RL) models. In DAVE, the optimizer is composed of a reinforcement learning agent, Q-learning [108]. At each time step t, the RL agent observes a set of metrics including the client playback buffer occupancy, past video encoding configuration decisions, and several network measurements (e.g., throughput). These values are defined as state s_t. The state will be fed in the Q-table, which outputs the action a_t (the video encoding configuration for the next interval). After applying the action, the state of the system transfers to s_{t+1} and the resulting QoE is then observed and passed back to the RL agent as a reward r_t. The RL agent uses the reward information to train and update its Q-table.

State: After the downloading of each frame t, the RL agent sends the state $s_t = (\vec{\overline{x}}, \vec{\overline{\tau}}, b_t, l_t)$ to the Q-table. $\vec{x}$ is the network throughput measurements for the past configuration interval; $\vec{\tau}$ is the downloading time of the past video frames within the configuration interval. The mean value of the throughout and downloading time of the whole configuration interval is used to overcome the frequent fluctuation of the network condition. b_t is the current buffer level of the player, and l_t is the currently used video encoding configuration.

Action: Each action is a video encoding configuration $a_t = (R_t, F_t, C_t, P_t)$. R, F, C, and P represent the frame resolution, frames per second, video compression rate, and video encoding speed, respectively. As mentioned earlier, there are millions of possible video encoding configurations. In this work, we only adopt the most commonly used values of each encoding parameter, as detailed in Table 3.2. Thus, there are 500

$(5 \times 4 \times 5 \times 5 \times 1)$ different configurations and the action space is limited to 500.

Some of these actions result in poor video quality. As shown in Fig. 3.6, all the actions in the lower-left corner will generate extremely poor quality video (the VMAF is less than 20). Moreover, there are many actions that result in close VMAF values but different encoding times with the same bitrate. To speed up the training speed of the RL agent, we eliminate these actions. We can also observe that there are many configurations result in close VMAF values but different encoding time with the same bitrate. We cannot determine which action is better until we evaluate it with the real-time video streaming system. In addition, We apply the super resolution model on all the videos encoded with down-scaled resolutions. Videos can always achieve the best video quality at different bitrate levels after applying super resolution.

3.4.4 Implementation

The video QoE and end-to-end latency determine the performance of real-time video streaming applications. Thus, the reward function of the RL agent is defined as:

$$\begin{aligned} Reward_n = \mu_0 VMAF^n - \mu_1 T_n - \mu_2 \left|VMAF^{n+1} - VMAF^n\right| \\ -\mu_3 (L^n_{encoding} + L^n_{trans} + L^n_{SR}), \end{aligned} \tag{3.10}$$

where n represents the video chunk index. Following the setting in [5, 104], the penalty parameters for rebuffering, smoothness, and latency μ_1, μ_2, μ_3 are set to 1.5, 0.02, and 0.005, respectively. μ_0 is set to 0.01 to balance the weight between the video quality and other QoE factors. The policy is updated as:

$$Q(s,a) \leftarrow (1-\alpha) \cdot Q(s,a) + \alpha \cdot (r + \gamma \cdot \underset{a'}{max} Q(s',a')). \tag{3.11}$$

To train the RL agent, we set the discount factor γ to 0.9. The learning rates for the RL model α is configured to be 0.001. We use the simulation platform provided by

ACM Multimedia 2019 Grand Challenge [104] to train the RL agent. The reason for using Q-learning instead of neural network based RL models (e.g., Deep Q-learning) is that our state space is composed of discrete values. Q-learning is a more efficient and lightweight algorithm for our proposed framework.

3.5 Performance Evaluation

In this section, we provide an extensive performance evaluation of the proposed dynamic video encoding configuration protocol.

3.5.1 Experiment Setup

Network traces: We use the network traces provided in [104] to train and evaluate DAVE. All the network traces are collected from a real network environment. There are three different network environments (low bandwidth, medium bandwidth, and high bandwidth network), and 40 traces for each network environment. Each network trace contains the network throughput with a duration of 2400 seconds, at a 0.5 seconds granularity. The average throughput of the three network environments is 1205 kbps, 1558 kbps, and 1949 kbps, respectively; the standard deviation of the network throughput is 0.53, 0.79, and 1.09, respectively.

Video traces: Since there are 500 actions in the RL agent, we encode a video into 500 different video tracks and collect 500 video traces. Each track is generated with a different encoding configuration. We use $ffprobe$ within FFMPEG to extract the data size of each frame in the generated video to generate the video traces.

Testbed: We build a real testbed to evaluate the performance of DAVE. The network is simulated with Mini-Net [109]. As shown in Fig. 3.7, there are total 30 nodes in the network and two nodes are bridged with real devices to implement the real-time video transmission functions. The link data rate is 50Mbps, and the delay of the links follows the normal distribution with $\mu = 6ms, \sigma = 1$. There are one video sender and one video receiver. The video sender is a Jetson Xavier and the video

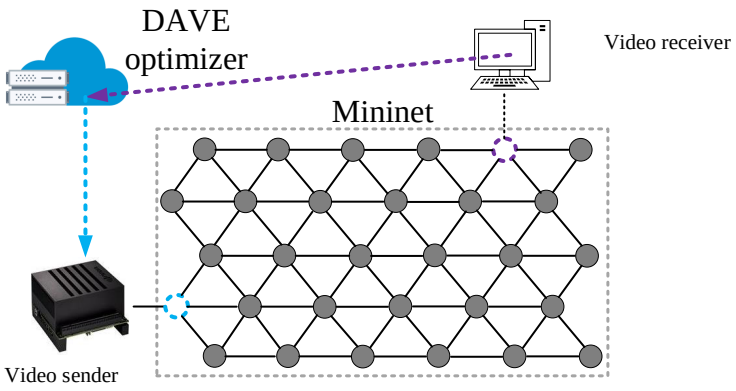


Figure 3.7: The topology of the testbed.

receiver is an Alienware Aurora R8 desktop. The DAVE optimizer is located in the video sender. It will collect the statistic information from the receiver. We adopt the super resolution model FSRCNN-S implemented with Tensorflow. An NVIDIA RTX 2020 is used to test the performance of the super resolution model.

Baseline algorithms: We compare DAVE with two other real-time video streaming algorithms:

- **Baseline**: directly sending the video frames captured by the camera, without any control.
- **GCC**: using several fixed strategies to configure the video frame resolution and FPS (following the same settings in [16]. For instance, reduce 1/3 resolution when the network throughput is lower than a certain level).

To further evaluate the impact of video configuration, we compare DAVE with three ABR algorithms: **BBA**, **MPC**, and **robustMPC** [12, 103]. We modified these three ABR algorithms to integrate with the real-time video streaming system.

3.5.2 Average Simulated System Performance

As shown in Fig. 3.8, we present the system performance (the sum of the rewards of all the video chunks) of each algorithm in each of the three network environments. For all of the networks considered, DAVE exceeds the performance of the other two

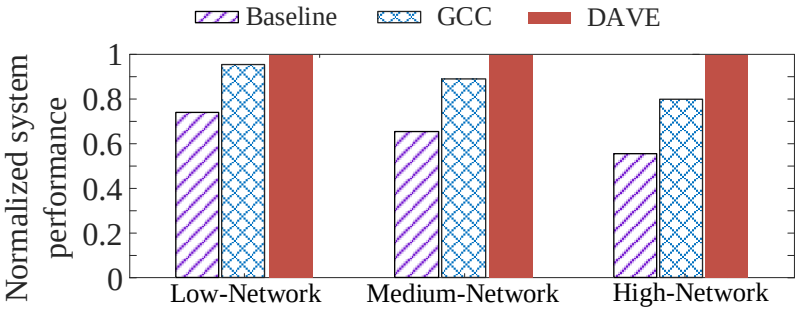


Figure 3.8: The normalized average system performance.

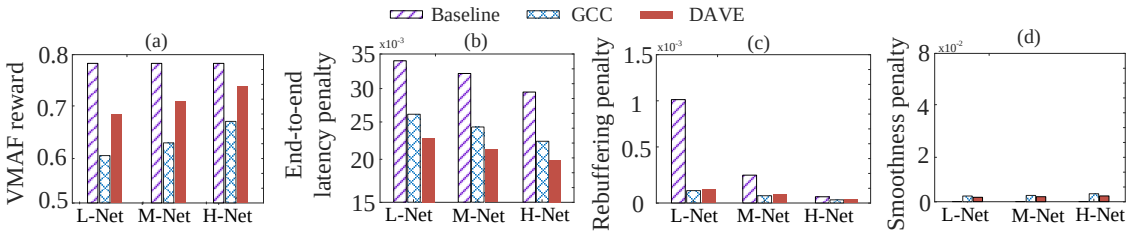


Figure 3.9: The average reward of different evaluation metrics in the simulation platform. (L-Net, M-Net, and H-Net refer to three different network environments as described in Section 3.5.1)

algorithms. DAVE outperforms GCC by 6%, 12%, and 20% in each network, respectively. The reason for only 6% enhancement in the low-bandwidth network is that all the three algorithms including DAVE will generate a large amount of low bitrate videos to avoid rebuffering. The dynamic adaptation of encoding configuration cannot achieve much improvement with such a low bitrate. However, DAVE can achieve higher system performance improvement with the increase of the average network bandwidth.

3.5.3 Evaluation of QoE Metrics

To better understand the QoE gains obtained by DAVE, we analyze the performance of DAVE on the individual terms in the reward definition in (3.10): the average gain of video quality (VMAF), the penalty from end-to-end latency, rebuffering, and smoothness.

Video quality: as shown in Fig. 3.9(a), with the help of the super resolution model, the video quality can be significantly improved. For all the three network en-

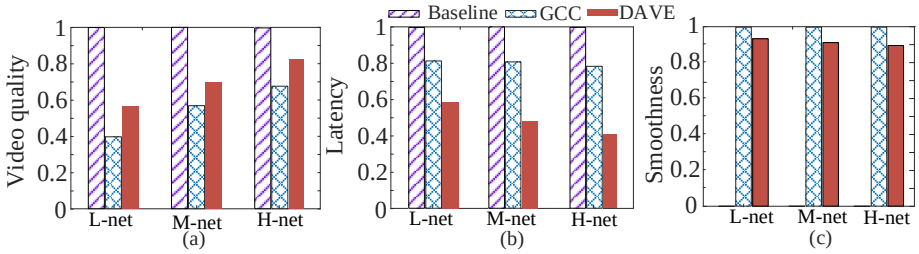


Figure 3.10: The average reward of different evaluation metrics in the testbed.

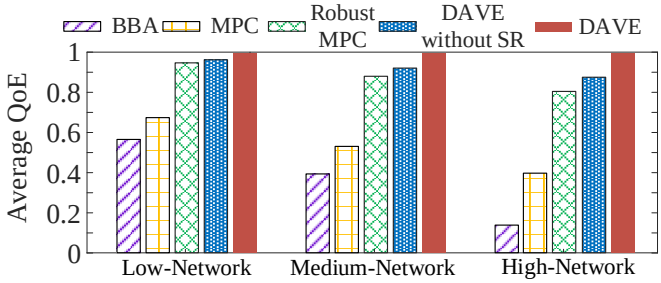


Figure 3.11: The normalized average system performance.

vironments, DAVE can improve the video quality by 14% on average compared with the GCC algorithm. The baseline approach uses the fixed video encoding configuration with the highest video quality, thus it can achieve the highest VMAF reward. However, this will bring poor performance of other evaluation metrics.

End-to-end latency: as shown in Fig. 3.9(b), the baseline approach brings the highest end-to-end latency penalty. DAVE can achieve a significant reduction in end-to-end latency compared with the other two approaches. Compared with GCC, 21%, 24%, and 26% of the end-to-end latency can be reduced for the three networks.

Rebuffering and smoothness: DAVE can achieve minor improvement on the rebuffering penalty compared with GCC. The reason is that GCC will reduce the video quality to fit the real-time requirement. Thus, it can achieve a high score on rebuffering. The baseline approach will use the same video from the start to the end of the streaming, thus the smoothness penalty is 0. DAVE can achieve better smoothness performance than GCC.

3.5.4 Average System Performance from the Testbed

We compare the performance of DAVE with the other two algorithms over a real testbed under three different network conditions.

Video quality: as shown in Fig. 3.10(a), with the help of super resolution model, the video quality can be significantly improved. For all the three network environments, DAVE can improve the video quality by 15% on average compared with the GCC algorithm.

End-to-end latency: as shown in Fig. 3.10(b), DAVE can achieve a significant reduction in end-to-end latency compared with all other algorithms. Compared with GCC, 19.1%, 19.5% 21.5% of the end-to-end latency can be reduced, respectively.

Smoothness: as shown in Fig. 3.10(c), the baseline approach can achieve the highest smoothness. DAVE can achieve 12% lower smoothness penalty than GCC on average.

In summary, a significant improvement on end-to-end latency can be achieved by DAVE, and DAVE can notably improve the video perceptual quality. Additionally, DAVE can either match or exceed the performance of the other approaches.

3.5.5 DAVE vs. Existing ABR Algorithms

To better understand the QoE gains obtained by the proposed dynamic video encoding configuration, we compare DAVE to several ABR algorithms that use fixed video configuration. As shown in Figure 3.11, for all the networks considered, DAVE exceeds the performance of all other ABR algorithms. The closest competing scheme is robustMPC. DAVE outperforms robustMPC by 3%, 8%, and 11% for the low, medium, and high bandwidth network, respectively. If we apply the super resolution model in DAVE, the system performance will be increased by 6%, 12%, and 20%, respectively. The reason for only 3% enhancement under the low bandwidth network is that all the ABR schemes including DAVE will choose a large amount of low bitrate

video chunks. The encoding configuration cannot achieve much improvement with such a low bitrate. DAVE can achieve better system performance improvement with the increase of the average network bandwidth. In summary, DAVE can either match or exceed the performance of existing ABR algorithms. Significant improvement on end-to-end latency and video quality can be achieved by DAVE.

CHAPTER 4: Proposed Pearl for UHD Video Delivery

In this chapter, we propose a cloud computing based deep compression framework named Pearl, which utilizes the power of deep learning to compress UHD videos. Pearl compresses UHD videos from two respects: the frame resolution and the colorful information. In pearl, an optimal compact representation of the original UHD video is learned with two deep convolutional neural networks (DCNNs): super resolution CNN (SR-CNN) and colorization CNN (CL-CNN). SR-CNN is used to reconstruct a high resolution video from a low resolution video while CL-CNN is adopted to preserve the color information of the video. Moreover, new channel-based super resolution models are developed to overcome the GPU memory shortage problem. In pearl, instead of applying the traditional RGB-based super resolution model, three separate super resolution models are trained based on the Y, U, and V channels of UHD videos. These super resolution models are used to reconstruct a UHD video from a low-resolution video. With Pearl, super resolution algorithms can be successfully applied to UHD videos. Pearl focuses on video content compression in two new directions. Thus, it can be integrated with any existing video compression system. With Pearl, the data size of UHD videos can be significantly reduced. At the same time, the efficiency of video encoding and decoding can also be improved with Pearl. To the best of our knowledge, Pearl is the first deep learning driven compression framework on UHD videos.

4.1 Problem Description

Ultra-high-definition (UHD) videos (4K and 8K) are enjoying increased popularity in people's daily lives because of the better visual experience. It will take account

for 22% of the whole network video by 2022 [8]. However, UHD videos will bring high pressure on data transmission and storage because the data size of 4k and 8k resolution are 4 times and 16 times of HD resolution for a video.

In recent years, there are many approaches that try to use deep learning techniques in the video delivery system [6, 49, 110]. As shown in Figure 4.1, there are main 4 steps in the deep learning driven video delivery system. The first step is to train the deep neural network (DNN) models for video compression. Because training the DNN model requires adequate computation resources, especially for UHD videos, this step can only be done on the cloud. Another limitation is that existing systems need to train (tuning) one DNN model for each video chunk to overcome the versatility problem, resulting in a large number of separate models for a long video. This brings additional storage and bandwidth cost for the video delivery system [111].

The next step is from the cloud to the edge servers which are closed to clients. Content Distributed Networks (CDNs) [2, 3] are the main tools that content providers like Google and Netflix use to improve the video delivery performance in this step. CDN consists of a group of servers that are placed across the world. These servers pull the contents from the cloud server and cache a copy of them allowing visitors to retrieve the content from the nearest server. CDN serves a large portion of the video deliveries across the Internet. One of the main challenges of CDN is the limited storage space of the distributed servers, which leads to that cached videos on the distributed servers will be frequently replaced [4]. This shortage will be amplified with the increasing amount of UHD videos in CDNs.

The third step is from the edge server to the client. The network bandwidth between servers and clients is the key factor to determine the user quality of experience (QoE) in this phase. Caused by the dynamic change of wireless networks, the user QoE suffers directly when the network throughput is low. Adaptive bitrate (ABR) algorithms [5, 6, 112, 113, 114] are widely used to optimize video transmission in this

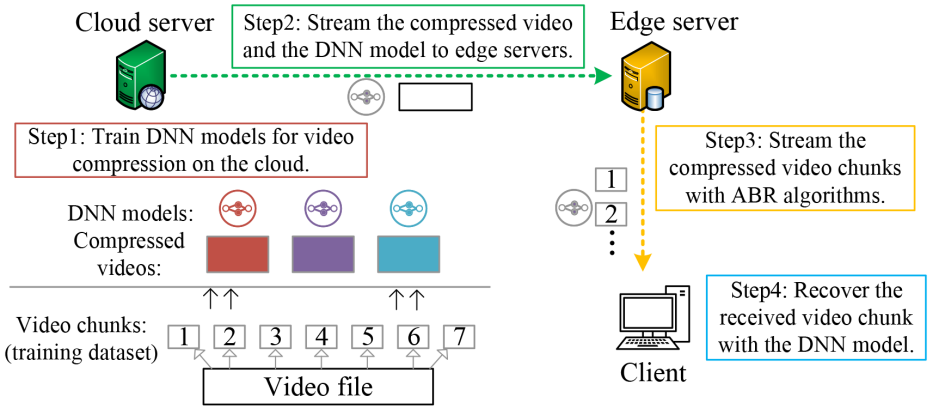


Figure 4.1: The video delivery system.

phase. There are already several advanced ABR algorithms that apply deep learning techniques. However, these works are mainly focused on improving the performance of existing ABR systems. The challenges brought by UHD videos are not solved.

In the last step, the compressed video can be recovered to the original video with DNN models on the client device. However, most of the existing DNN models are trained on HD videos. When the model is trained on UHD videos, a challenge is that the GPU memory required to execute the model is significantly increased, such a high requirement cannot be satisfied by most of the existing GPU models on the client device.

4.2 Research Motivation and Challenges

The performance of the video delivery system has been greatly improved by employing CDN and ABR algorithms. However, current frameworks will face the following challenges when UHD videos are becoming more and more popular:

- Since the data size of UHD videos is 4-16 times larger than that of HD videos, there will be frequent replacements of cached videos due to the limited storage space on the distributed servers. Thus, the efficiency of CDNs will be decreased. Meanwhile, the large video data size also demands high network bandwidth. As a result, low bitrate video chunks with a low video quality are frequently selected

in the ABR system, which significantly degrades user QoE.

- With the power of deep learning techniques, many new algorithms can be used to improve the image and video quality, such as the super resolution algorithm. However, the GPU memory will become the main obstacle in applying these algorithms to UHD videos.

- Existing systems train separate DNN models for each video trunk to overcome the versatility problem of deep learning algorithms. However, transmitting such a large number of models brings additional storage and bandwidth cost.

To address above challenges, we propose Pearl, a system that applies deep learning techniques on video content compression to maximize video delivery performance and user QoE. To tackle the versatility problem in the first step, two deep convolutional neural networks (DCNNs) are designed and trained in Pearl to learn an optimal compact representation from an input video, which preserves the structural information and color information. These two DCNNs can be widely applied to different types of videos according to the evaluation results. After the compression, the video data size is significantly reduced. We can recover high-quality videos with decompression using the DCNN models. In this case, a large amount of network bandwidth resources and storage spaces used for video delivery are reserved. Meanwhile, the transmission latency will also be reduced. As a result, the challenges brought by UHD videos in steps two and three are solved. To overcome the GPU memory shortage problem in step four, we design and apply channel-based super resolution models. Note here, Pearl focuses on video content compression from two new directions: resolution and color channel. As a result, it can be integrated with existing ABR systems and video encoding algorithms.

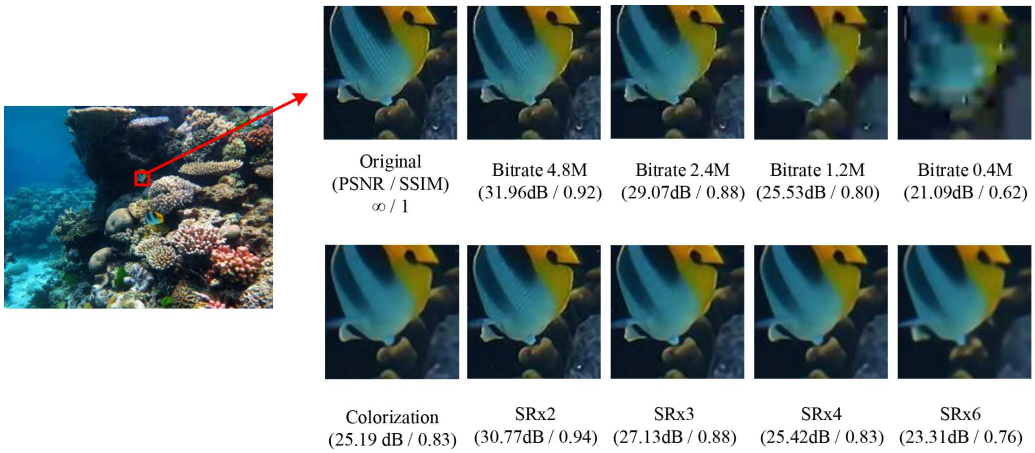


Figure 4.2: The visualization results of SR and CL models.

4.3 Video QoE Study

In this section, we show a comprehensive study of the video quality in current video delivery systems. According to experiment results, our proposed framework can achieve similar video quality with a smaller data size compared with existing algorithms. Peak signal-to-noise ratio (PSNR) and the structural similarity index (SSIM) are used to evaluate the video frame quality in this paper. At first, we evaluate the video frame quality of the adaptive bitrate video system. A 2k HD video is encoded with 5 bitrates: 6000k, 4800k, 2400k, 1200k, and 400k. H.264 is the video codec used here. The 6000k bitrate video is defined as the original video. The PSNR and SSIM of different bitrate frames are shown in the first row of Figure 4.2. We can find that the PSNR and SSIM keep decreasing with the decrease of bitrate. When the bitrate is less than 1000kbps, the frame is blurry.

Then, we study the impacts of super resolution and colorization algorithms on video quality. We choose a state-of-the-art super resolution and a colorization model to perform the experiments. For the super resolution model, we choose 4 different scales:{x2,x3,x4,x6}. After applying the super resolution models with different scales, we measure the PSNR and SSIM of the generated frames. The results are presented in the second row of Figure 4.2. Compared with adaptive bitrates, the results generated

by the super resolution model are smoother than the frames with different bitrates from the visual experience. The super resolution model can always achieve higher SSIM and lower PSNR values. When it comes to the low bitrate, super resolution can achieve better video quality. The performance of the colorization is close to the performance of SRx4. To improve the performance of the colorization model, we adopt a reference frame based colorization algorithm. More details are shown in Section 4.4.2. Another factor we investigate here is the frame data size. As shown in Table 4.1, the frame data size of the super resolution and colorization algorithms are much less than the ABR algorithms.

Table 4.1: The comparison of frame data size

Bitrate	Data Size	SR	Data Size	CL+SR	Data Size
4800k	5.8MB	x2	1.5MB	x2	491KB
2400k	5.7MB	x3	687.5KB	x3	227KB
1200k	5.3MB	x4	399.5KB	x4	132KB
400k	3.8MB	x6	185.9KB	x6	62KB

The results of the video quality comparisons prove that we can use the super resolution and colorization models to compress the video content. The video quality will not be decreased while the data size of the video will be significantly reduced.

However, we fail to directly apply the existing super resolution model (MDSR [51]) to UHD video frames for all the scales. The frame resolution of 4k and 8k videos is 3840x2160 and 7680x4320, respectively. As shown in Figure 4.3 (a), the GPU memory required for 4k and 8k video frames are 9 GB and 13 GB when the scale is set as x4, which cannot be supported by most of the existing GPU models. Such high-resolution frames cannot be directly generated by the deep compression models. To solve this problem, we crop the 4k and 8k video frames into multiple parts. The 4k video frame is divided into four parts and each part is a 2k sub-frame. The 8k video frame is divided into 16 sub-frames. We then can successfully apply the super resolution model by frame cropping. However, the model execution time will

be significantly increased. As shown in Figure 4.3 (b), it takes over 20 seconds to process an 8k video frame. At the same time, image cropping and combining cause additional frame processing time.

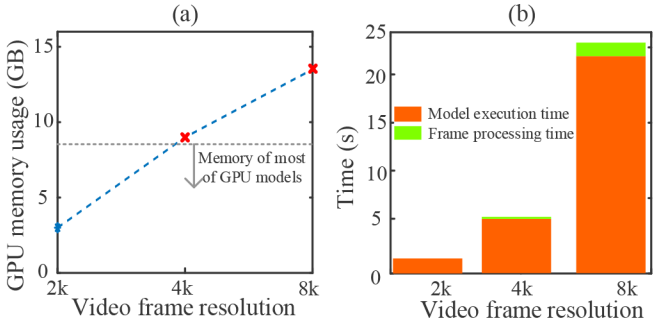


Figure 4.3: The GPU memory usage and time consumption.

We can conclude that super resolution algorithms can achieve good performance on video compression. However, existing systems face many challenges when super resolution algorithms are directly applied to UHD videos. Thus, we propose Pearl, the first deep learning driven compression framework for UHD videos.

4.4 Proposed Design and Implementation of Pearl

In this section, we detail the design and implementation of Pearl, a system that applies the super resolution and colorization algorithms for video compression. First, we describe the whole system framework. Then, the design of the colorization and super resolution algorithms are presented. After that, we present the studies of the versatility problem of super resolution and colorization models. Finally, we explain the implementation details of the deep learning models.

4.4.1 System Framework

Pearl mainly contains two parts: encoder and decoder. As illustrated in Figure 6.2, the original video frames will be encoded with existing encode algorithms (e.g., H.264) to compress the video data size before applying our proposed deep compression models. Then the encoded video will be deeply compressed from a high resolution colorful

video to a low resolution gray video with our encoder. The deeply compressed video will be reconstructed to the high resolution colorful video with the decoder. The detailed steps are shown in Figure 4.5.

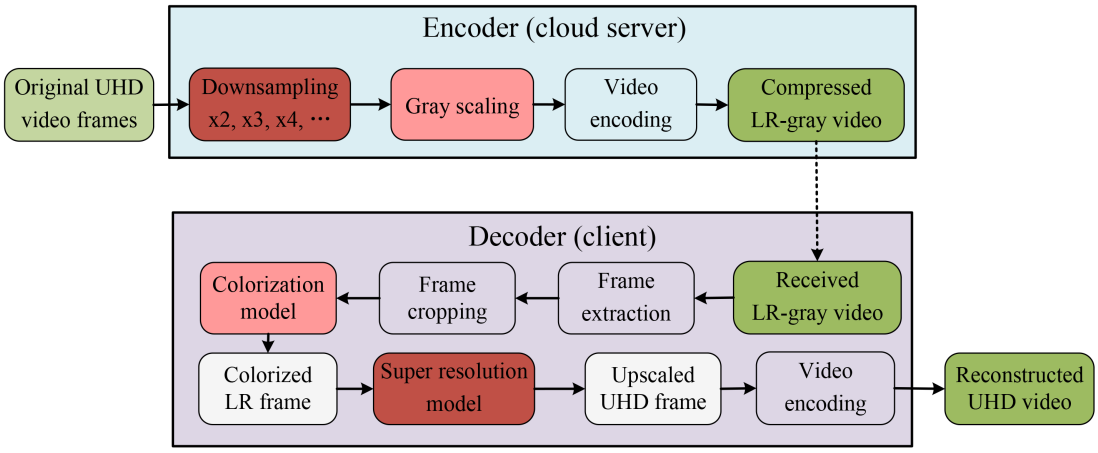


Figure 4.4: The system framework.

Encoder: The video on the cloud server will be down-sampled and gray-scaled from high resolution colorful videos to low resolution gray videos. The deeply compressed video will be distributed to CDNs. Training a deep learning model normally takes dozens to hundreds of hours. If the trained model cannot achieve good performance on different types of video, it will be very challenging to widely adopt the deep compression frameworks. The reason is that we need to train an individual deep learning model for each kind of video or train separate models for each chunk of a video, which is not practical, even with the help of fine-tuning techniques. To solve this problem, we show a comprehensive study of the model performance on different types of video content. According to experiment results, the super resolution model trained on a large dataset performs well on different types of videos. However, the trained colorization model has a poor performance. To solve the versatility problem of colorization models, we propose a reference-based colorization model.

Decoder: After receiving the deep compressed video from the edge server, we can reconstruct the video from low resolution gray video to high resolution colorful

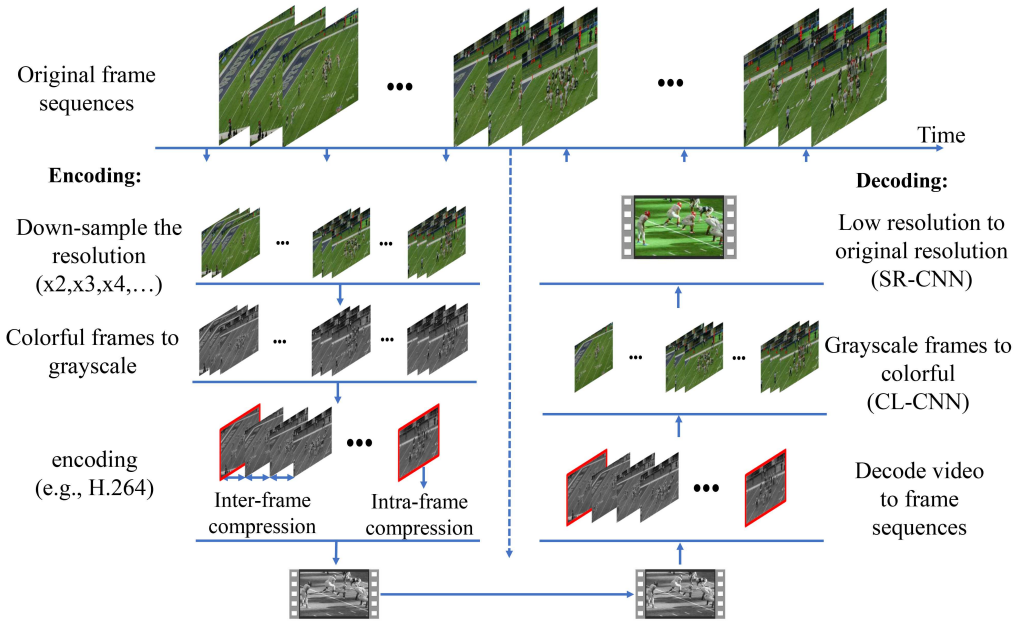


Figure 4.5: The video encoding scheme.

video. The colorization model is applied before the super resolution model. The reason is that the colorization model acquires more GPU memory compared with super resolution models. In addition, the time complexity of the colorization model is also higher than the super resolution model. Thus, applying the super resolution model to low resolution colorful video is more efficient than applying the colorization model to high resolution gray video. To solve the GPU memory shortage problem, we propose channel-based super resolution models.

4.4.2 Deep Learning Model

As shown in Figure 4.6, the deep learning model consists of two DCNNs: SR-CNN and CL-CNN. At first, the low resolution (LR) gray frames will be fed into the CL-CNN model. LR colorful frames will be generated. Then, the SR-CNN will convert the LR colorful frames to high resolution (HR) colorful frames.

SR-CNN: There are several state-of-the-art super-resolution algorithms, such as EDSR and ESRGAN [51, 52]. The input of most existing super resolution algorithms

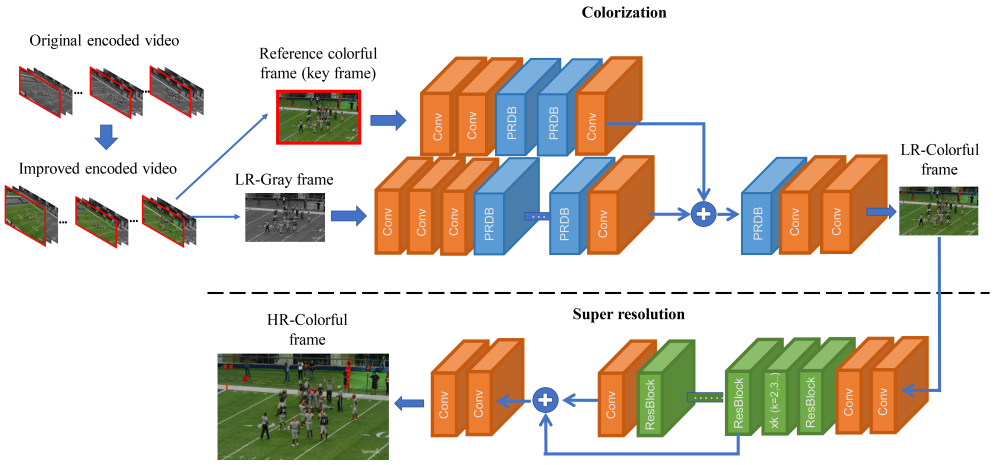


Figure 4.6: The deep compression model.

is an RGB video frame. The main obstacle of adopting these algorithms on UHD video frames is the limited GPU memory. To perform super resolution on UHD video frames, we propose channel-based super resolution models. As shown in Figure 4.7, instead of using the whole RGB video frame as the input to the super resolution model, each separate frame channel is set as the input. In this case, the data size of the input of the super resolution model is reduced by 3 times. As a result, the GPU memory shortage problem is solved. We also present the pipeline of the baseline approach (CropSR) in Figure 4.7. In CropSR, all video frames are extracted from the received colorized LR video. These RGB image frames are cropped into several subframes. After applying the super resolution model to each subframe, the generated results will be combined to generate the upscaled UHD frames. The main advantage of channel-based SR model is that it only needs to execute 3 times to generate a UHD video frame. However, 4 executions for a 4k video frame and 16 executions for an 8k video frame are needed in CropSR.

Another challenge of adopting state-of-the-art SR models is the time complexity. For example, the inference speed of EDSR for 1020x768 frames is 2.08 frames per second, which makes it impossible to achieve real-time video processing, especially for UHD videos because the inference speed is directly affected by the resolution of the

input frame. To meet the real-time constraint for the high resolution frame, we adopt NAS-MDSR (a downscaled SR model), which uses scalable DNN to enable anytime prediction [6]. With NAS-MDSR, the inference speed can reach 31.34 FPS (Within 1 second, 31 frames can be generated with the super resolution model). The normal FPS for UHD videos is 30 FPS. Thus, a real-time frame super resolution process can be achieved. For each scale of super resolution (x2,x3,x4,...), we need to train for separate SR-CNN models.

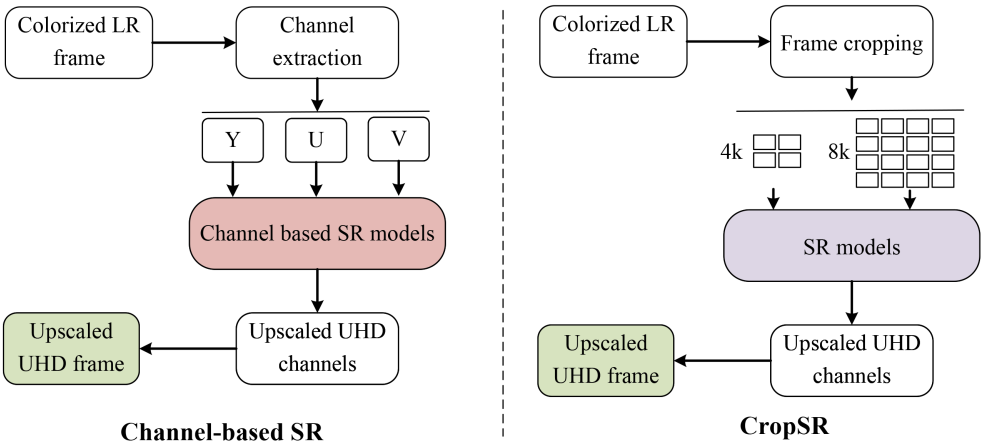


Figure 4.7: The framework of super resolution models.

CL-CNN: Colorization is an even more complicated task compared with the super resolution because of the large learning space of colorful information. We adopt Pix2pixHD [115], a state-of-the-art colorization algorithm, as our base CL-CNN model. According to experiment results, the video quality (PSNR and SSIM) of the colorization model based on Pix2pixHD is not robust. For instance, the PSNR and SSIM of some frames are 20.55 dB and 0.63, which is worse than the quality of the smallest bitrate video in the adaptive bitrate encoding approach. To improve the performance of the colorization model, a dense encoding pyramid network (DEPN) is adopted [116]. DEPN is a reference frame based colorization algorithm. The main difference is that the keyframes will keep the colorful information instead of converting to gray frames when we encode the video. DEPN will use the colorful keyframes as the

reference frame to colorize the gray frames. The colorful keyframe will provide guiding information for colorization. Thus, the performance of the colorization model is improved. Another advantage brought by DEPN is that the colorization model training by DEPN can be used on the colorization of multiple videos. For Pix2pixHD based colorization model, we need to train an individual colorization model for each video. The objects and scenes keep changing in videos. If we only have a few colorful keyframes, the coloration performance will be poor. A proper interval between the key colorful frames needs to be chosen.

Table 4.2: The performance of DCNN models

-	SR (x4)	Channel-based SR(x4)	CL	Reference-based CL
PSNR	33.90	32.62	25.47	32.77
SSIM	0.87	0.83	0.72	0.91

4.4.3 Model Versatility

To study the model versatility problem of super resolution and colorization models, we evaluate the performance of our proposed models with 27 video clips from 9 different categories that are randomly downloaded from the Internet. As shown in Table 4.2, the average PSNR and SSIM of the original SR model is 33.90 and 0.87. The proposed channel-based SR model has a slight performance degradation. However, the performance of both of these two SR models can be classified as good [117]. In this case, we can conclude that the SR model can be widely adopted on different types of videos. Training a separate SR model for each video chunk or each video is not necessary. For the CL model, the average recovery performance is poor. The reason is that there is no guiding information for colorizing the gray image. For instance, a vehicle with $color_a$ in the first video chunk may be very similar to another vehicle with $color_b$ in the second video chunk when video frames containing these two vehicles are converted to gray images. The colorization model fails to correctly

colorize these two vehicles at the same time. Therefore, one colorization model cannot perform well on multiple video chunks and different types of videos. In this paper, we propose a reference-based colorization model to overcome the versatility problem of the colorization model. As shown in Table 4.2, the recovery performance of the reference-based CL model is significantly improved as compared with the original CL model and it can even achieve better recovery performance on SSIM as compared with SR models, which indicates that it can be widely adopted on different types of videos.

We use popular DNN architectures as the backbone of our proposed DNN models. Instead of designing new models, we focused on solving the challenges of applying existing models on UHD videos in this work (directly applying the existing state-of-the-art DNN models fails to address these challenges): the channel-based super resolution model is designed to solve the GPU memory shortage problem and the reference-based colorization model is designed to tackle the model versatility issue. These two models have never been studied in existing video delivery systems.

4.4.4 Implementation

Because there are no 4k and 8k video data sets available for training our super resolution model, we use the most widely used 2k video data set, DIV2k and Flick2k [118], to train the proposed channel-based super resolution models. The NAS-MDSR (SR-CNN) and Pix2pixHD (CL-CNN) are implemented with Pytorch. For training the SR-CNN model, the frames (1920x1080) are downsampled to {960x540, 640x360, 480x270, 320x180} for {x2, x3, x4, x6}. For the hyperparameters, we adopt the same setting as in [6]. The batch size is 64, and the learning rate is 10^{-4}. For fine-tuning the Pix2pixHD model, we extract 1 frame per second as the training dataset of each video. The video frames (1920x1080) are randomly cropped into 512x512 pixels. Then, these cropped frames will be converted into gray images as the training dataset. The batch size is 8, and the learning rate is $2x10^{-4}$. The improved colorization algorithms DEPN

is based on Caffe [119]. The network was trained with the learning rate of $3x10^{-5}$, and batch size is 5. The optimization algorithm applied in all three deep learning models is *Adam*.

4.5 Performance Evaluation

In this section, we provide an extensive evaluation of the proposed video deep compression framework under two phases of experiments. The first phase experiments are performed with the video delivery system. We integrate the deep compressed videos with the state-of-the-art ABR algorithm in the second phase experiments. The main findings are:

- **Video QoE:** Pearl can further compress the video data size from 84% to 97% with existing video encoding algorithms. Compared with existing adaptive bitrate encoding algorithms, we can reduce 65% of the video data size with the SR model, and 84% with the CL+SR model. Considering the video visual quality, Pearl can improve the PSNR and SSIM (27%, 13%) and (10%, 11%) for the SR model and the CL+SR model, compared with the adaptive bitrate encoding algorithm.

- **Network transmission latency:** We perform the video delivery experiments under three different network conditions. On average, the network latency can be reduced by 66%-95% after applying the SR model. 73%-95% of network latency can be reduced with the CL+SR model.

- **ABR system:** After integrating Pearl with the state-of-the-art ABR algorithm, we improve the average QoE of the ABR system from 0.62 to 1.47 and 1.52 for the SR model and the CL+SR model, respectively.

4.5.1 Methodology

Video: The HD and UHD videos for experiments are downloaded from YouTube. For HD videos, we download videos from 9 popular categories (1: Beauty, 2: Comedy,

3: Cook, 4: Entertainment, 5: Game, 6: Music, 7: News, 8: Sports, 9: Technology). For each of the nine Youtube channel categories, we download three videos. For UHD videos, we download 2 videos for 2 categories(1: Beauty, 2: Sports) because of the limited UHD video resource online. All the videos are cut into 5 minutes length and re-encoded. The video processing library and encoding codec we used is FFMPEG and H.264, respectively. The encoding frame resolution, frame chunk size (GOP), and the frame rate are:

- 2k videos: {1920x1080, 4 seconds, 24 FPS}
- 4k videos: {3840x2160, 4 seconds, 30 FPS}
- 8k videos: {7680x4320, 4 seconds, 60 FPS}

Network: We test the performance of the video delivery under three different network environments:

- $Local - LowBW$: local low bandwidth wireless network. It is composed of one router and two workstations, one is the transmitter and another one is the receiver, the average bandwidth is 3 Mbps.
- $Local - HighBW$: local high bandwidth wired network. The videos are stored on the cloud server. We download the videos with a workstation, the average bandwidth is 108 Mbps.
- $Remote - MediumBW$: remote network. We put videos on a remote server, and fetch videos from a local workstation. The average bandwidth is 7 Mbps.

ABR algorithm and network trace: There are several widely used ABR algorithms, such as BOLA, MPC, robustMPC [60, 12]. However, Pensieve outperforms all of these algorithms. NAS is another state-of-the-art ABR algorithms. Because the code of NAS is not released yet, only Pensieve is used to integrate with the proposed

deepcompress system during the experiments. We use the HSDPA network trace dataset provided by [5] to evaluate the performance of the integrated system. There are total 142 network traces in the dataset. The bitrates used for adaptive bitrate encoding are (300, 750, 1200, 1850, 2850) kbps, (1, 4, 8, 14, 20) Mbps, (4, 12, 24, 48, 60) Mbps for 2k, 4k, and 8k videos, respectively.

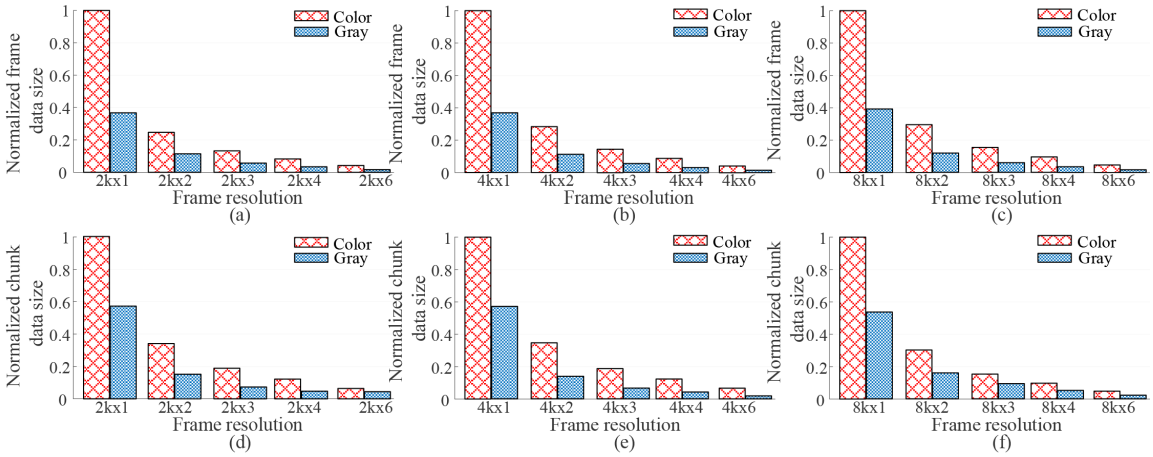


Figure 4.8: The normalized video frame and chunk data size.

Table 4.3: The comparison performance of UHD video frames

Bitrate (Mbps)	Size (Mbps)	PSNR	SSIM	SR	Size (Mbps)	PSNR	SSIM	CL+SR	Size (Mbps)	PSNR	SSIM
60	36.49	∞	1	8kx1	36.49	∞	1	8kx1	11.97	-	-
48	33.65	35.40	0.91	8kx2	10.12	44.43	0.98	8kx2	3.69	37.09	0.97
24	31.97	32.97	0.87	8kx3	5.11	42.42	0.97	8kx3	1.81	39.95	0.98
12	29.86	30.15	0.81	8kx4	3.07	39.51	0.96	8kx4	1.07	38.20	0.96
4	28.14	24.91	0.67	8kx6	1.45	34.76	0.91	8kx6	0.5	31.24	0.87
-	-	-	-	average gain	**84.63%**	**31.19%**	**18.38%**	average gain	**94.5%**	**19.15%**	**16.9%**
20M	9.03	∞	1	x1	9.03	∞	1	x1	3.23	-	-
14M	8.87	40.31	0.96	x2	2.49	49.23	0.99	x2	1.27	34.77	0.96
8M	8.60	38.00	0.94	x3	0.98	42.90	0.98	x3	0.48	37.92	0.98
4M	8.09	34.43	0.90	x4	0.76	39.36	0.96	x4	0.28	35.08	0.96
1M	7.00	25.77	0.76	x6	0.36	35.85	0.92	x6	0.13	30.57	0.89
-	-	-	-	average gain	**86.5%**	**23.32%**	**8.77%**	average gain	**93.7%**	**2.66%**	**7.01%**

Experiment settings: To evaluate the video QoE and network latency, each 2k video is encoded into 6000 Kbps. The 6000 Kbps videos are set as the original videos for down-scaling and gray-scaling. We choose 4 scales {x2,x3,x4,x6} to downscale the frame resolution. The downscale method is bicubic. Then, we convert each down-scaled video to gray-scale video. For UHD videos, each video is encoded into 20 Mbps and 60Mbps for 4K and 8K videos. These videos are set as the original video for down-scaling and gray-scaling. The down-scaling and gray-scaling approaches are

the same with 2k videos. To satisfy the requirements of the training of deep learning models, we use a server with 8 GTX 1080TI GPUs and a workstation with 3 GTX TITAN X. A computer with an RTX 2080TI GPU is used to execute all the trained models. The SCP function in SSH is applied to transmit the encoded videos.

QoE metrics: We have two different QoE metrics to evaluate the performance of Pearl: Video QoE and ABR QoE. Video QoE is used to evaluate video compression quality. There are three types of metrics: $size_f$ (frame data size), PSNR and SSIM (frame perceptual quality), and $latency_n$ (network transmission latency). The ABR QoE defined in Pensieve is used to evaluate the ABR system, which is defined as following:

$$QoE_{ABR} = \sum_{n=1}^{N} q(R_n) - \mu \sum_{n=1}^{N} T_N - \sum_{n=1}^{N-1} \left| q(R_{n+1}) - q(R_n) \right|, \tag{4.1}$$

where R_n represents the bitrate of chunk n. $q(R_n)$ is a function to calculate the video quality of video chunks n at bitrate R_n. T_N is the rebuffering time caused by downloading chunk n at bitrate R_n. The quality difference between two neighboring chunks is used to punish the changes of bitrate. Frequently changing of the bitrate will affect the smoothness of the video.

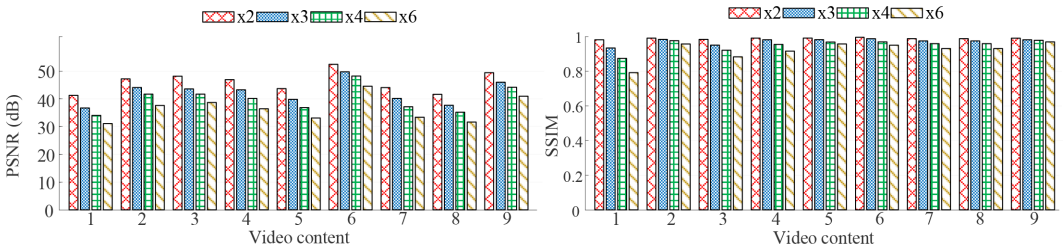


Figure 4.9: The performance of the SR models on 9 types of 2k videos.

4.5.2 Video QoE

To evaluate the video QoE, we have two granularities: frame and video chunk. The average frame data size of a video is less than that of an isolated image frame.

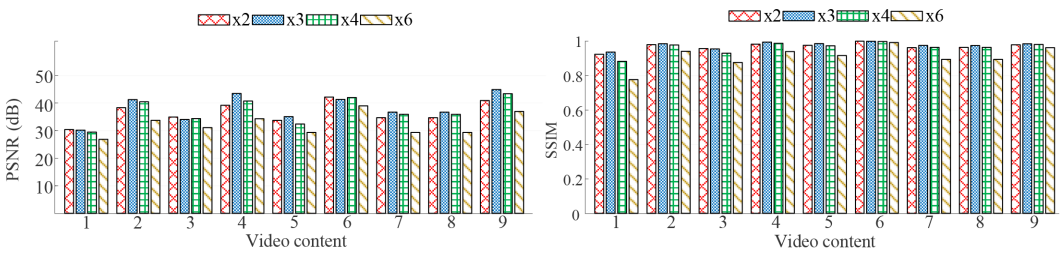


Figure 4.10: The performance of combined CL+SR models on 9 types of 2k videos.

The reason is that inter-frame compressing is applied by video encoding algorithms. We use video chunk level to evaluate the performance of combining Pearl with a video encoding algorithm. Nevertheless, videos are sent frame by frame in some scenarios, for instance, in augmented reality (AR) applications. The video inter-frame compression cannot be applied in these applications. Thus, we evaluate the performance of Pearl in image frame level compression.

Frame: Figure 4.8(a), (b), (c) show the data size of 2k, 4k, and 8k image frames, respectively. For 5 different down-sampling scales, gray-scaling can reduce the frame data size by 58.56%, 62.69%, 61.46% on average, for 2k, 4k, and 8k image frames, respectively.

Video chunk: The video chunk data size of 2k, 4k, and 8k videos are shown in Figure 4.8(d), (e), (f). Gray-scaling can reduce the data size of video chunks by 50.10%, 62.23%, 53.35% on average, for 2k, 4k, and 8k videos, respectively.

In summary, applying the SR model can reduce 70% to 95% of the frame data size and 65% to 95% of video chunk data size for UHD videos. 88% to 98% of frame data size and 84% to 97% of video chunk data size for UHD videos can be reduced with the CL+SR model. Down-scaling can significantly reduce the frame data size by reducing the pixel dimension of video frames. However, the frame visual quality is also decreased. If we recover the frame using up-sampling methods, the frame visual quality is too poor to satisfy the user visual requirement. Super resolution models can achieve much higher frame visual quality as compared with up-sampling

methods. With our proposed channel-based super resolution model, UHD videos can be down-sampled and recovered without losing much visual quality.

4.5.3 Performance of DCNNs

As shown in Figure 4.9 and Figure 4.10, we show the PSNR and SSIM of the reconstructed video frames for 9 types of 2k videos. There are two different approaches: only applying the SR model (SR-only) and combining the CL model with the SR model (CL+SR). According to the experiment results, the CL+SR approach brings more loss on PSNR. However, the CL model can help to keep the structure information of the original frame because of the reference frame structure in the colorization algorithm, which can make it achieve higher SSIM values.

We define the bottom-line construction quality of a video frame as (30 dB, 0.85) for PSNR and SSIM. The cumulative distribution function (CDF) graphs of the PSNR and SSIM are shown in Figure 4.11. For the SR-only approach, about 99% reconstructed frames are above the bottom-line quality for the x2 scale. 97%, 79%, 71% for x3, x4, and x6 scale, respectively. For the CL+SR approach, there are about 85% reconstructed frames which are above the bottom-line quality for the x2 scale, and 88%, 78%, 59% for x3, x4, and x6 scale, respectively. Here, the SR model and the CL model are not fine-tuned with the frame dataset of each video. This proves that the SR and CL models can be widely used on different types of videos without training individual models for each video. In this case, we do not need to transmit the deep learning model with the video, which results in lower network latency. There are some reconstructed frames with extremely high PSNR and SSIM values. The reason is that these frames contain a large percentage of pure color blocks. The SR and CL models can achieve high accuracy for recovering pure color blocks. When we combine the SR model with the CL model, we can find that the x3 scale can achieve higher performance than the x2 scale. The reason is that the CL model can only achieve poor performance on large resolution frames (e.g., x2 frames).

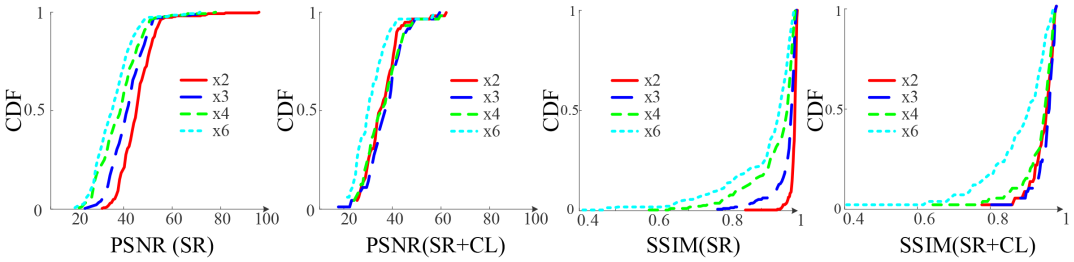


Figure 4.11: The UHD video frame reconstruction performance.

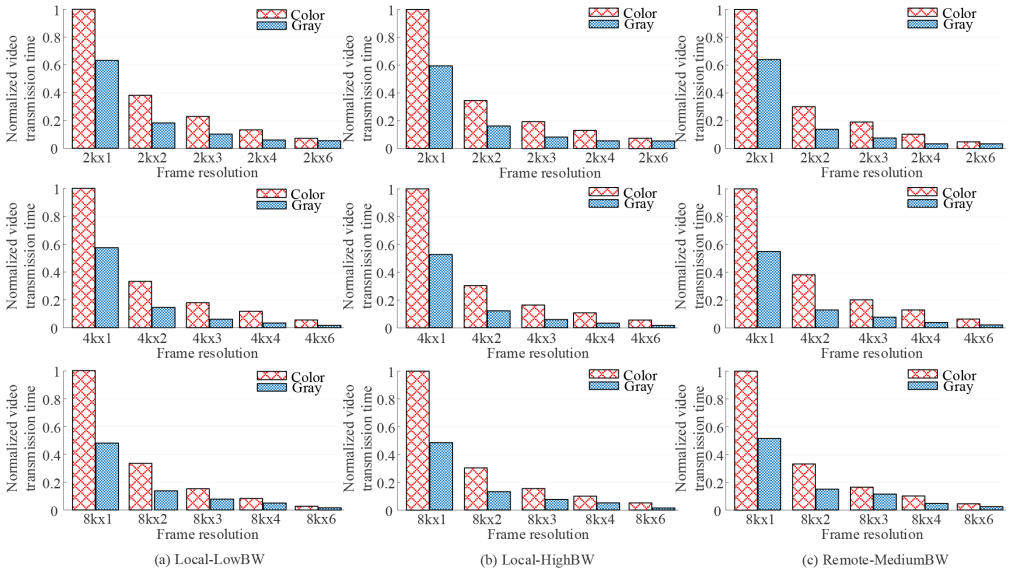


Figure 4.12: The network latency.

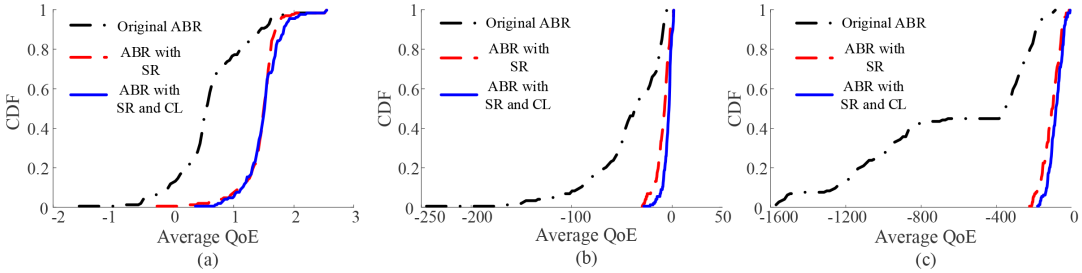


Figure 4.13: The performance of integrated system.

The reconstruction performances of the SR model and CL model for 4k and 8k video frames are presented in Table 4.3. We compare Pearl with an adaptive bitrate encoding algorithm. For the SR-only approach, we can save 84.63% and 86.5% of data size on average for 4k and 8k videos. The PSNR and SSIM are improved by (31.19%, 18.38%) and (19.15%, 16.9%) for 4k and 8k videos. For the CL+SR approach, we

can save 94.5% and 93.7% of data size on average for 4k and 8k videos. The PSNR and SSIM are improved by (23.32%, 8.77%) and (2.66%,7.01%) for 8k and 4k videos.

In summary, Pearl can improve (27%, 13%) and (10%, 11%) PSNR and SSIM of UHD videos with the SR model and the CL+SR model. We can conclude that Pearl can achieve better video quality (higher PSNR and SSIM values) with a smaller frame data size compared with an adaptive bitrate encoding algorithm for both the SR-only approach and the CL+SR approach. There are many trade-offs between SR-only approach and CL+SR approach. For instance, applying the colorization model costs more frame recovering time. However, the transmission time and the storage space are reduced, and the frame visual quality can also be enhanced by the colorization model. In this case, the ABR algorithm needs to be more adaptive to select the optimal video chunk. Traditional ABR algorithms only consider the tradeoffs between the bitrate of the video chunk and the transmission latency. However, the reinforcement learning based ABR algorithm adopted in our system can learn the relationships among all these factors and predict the optimal selection option.

4.5.4 Network Latency

With Pearl, the video data size will be reduced from 79% to 97% compared with original videos. A large amount server storage spaces are saved. Meanwhile, the video transmission latency will also be significantly reduced. We use three different networks to transmit 2k, 4k, and 8k videos, the network latency results are shown in Figure 4.12. Applying the SR model can reduce 66.68%-95.89%, 69.70%-94.54%, 64.47%-94.68%, of network latency for Local-LowBW, Local-HighBW, and Remote-MediumBW network, respectively. 70.07%-96.18%, 75.83%-95.51%, and 75.75%-95.02% of network latency can be reduced with CL+SR model. On average, 66%-95% and 73%-95 of network latency can be saved with the SR model and the CL+SR model respectively. The network latency is defined as the end-to-end video transmission time, which includes the model execution time.

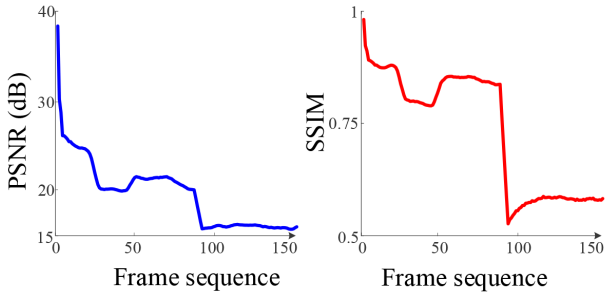


Figure 4.14: The performance of improved colorization model.

4.5.5 Pearl with ABR System

To evaluate the performance of integrating Pearl with ABR systems, we combine Pearl with Pensieve together. The result of applying the integrated system for 2k, 4k, and 8k videos are shown in Figure 4.13 (a), (b), and (c), respectively. We can find that the integrated system outperforms the original ABR system. Moreover, the CL+SR model outperforms the SR model 25%. All the QoE of 4k and 8k videos are negative numbers. The QoE of UHD video delivery suffered under the network simulated from the network trace. This proves that applying the deep compression framework to improve the QoE of UHD video delivery is necessary.

4.5.6 Performance of the Improved Colorization Model

In Figure 4.14, we show the performance of the improved colorization model. The PSNR and SSIM are decreasing with the increase of the frame sequence index, which is caused by the scene changing in the video. The average PSNR and SSIM of the Pix2pixHD colorization model are 26.14 and 0.86. The reference-based colorization model outperforms the Pix2pixHD model within 10 continuous frames. If we set a keyframe for every 24 frames, the lowest PSNR and SSIM are 22.01 dB and 0.84. The main advantage of the improved colorization model is that we can share a colorization model for a variety of videos instead of training an individual colorization model for each video. A large amount of training time is reserved. Moreover, it is unnecessary to transmit it with the video content during the video delivery.

4.6 Discussion

Integration of Pearl and ABR: In this work, the deep learning models we trained are only used for video content compression. They are independent of existing video compress algorithms and ABR algorithms. However, there is another approach to integrate the deep learning models and the ABR algorithms, which applies joint-training on the video compression models with ABR models. For instance, NAS integrates a super resolution model into a state-of-the-art ABR algorithm that uses a deep reinforcement learning model. The deep reinforcement learning model is trained with the effect of the super resolution model. The advantage is that the deep learning model in the ABR system can make better decisions compared with separate training. In this paper, we only focus on compressing the UHD video data size and reducing the network transmission latency of UHD videos in the CDN.

Super resolution or colorization: With super resolution model and colorization model, 2*K kinds of compressed videos will be generated. K is the number of down-sampling scales. Compared with the SR model, the CL+SR model can improve 39% and 18% on the frame and chunk data size compression, and the performance on PSNR and SSIM will be reduced by 12.8% and 1.33% respectively. If the network throughput is high, we can only use the SR model during the network transmission to obtain better video quality. When there are limited network resources, the CL+SR model can be adopted. Both the SR model and the CL+SR model outperform the existing adaptive bitrate encoding methods.

Interval of colorful keyframes: For the improved video colorization model, the keyframe keeps the color information instead of being converted to a gray frame. How to set the interval between keyframes needs to be carefully considered. If the interval of the keyframes is too small, the power of the colorization model is not utilized. If the interval between the keyframes is large, the performance of the colorization model will be poor. Video scene recognition algorithms can be applied to solve this problem.

It can detect the time of scene changing. When the scene changes frequently, we can choose a small keyframe interval, and vise versa.

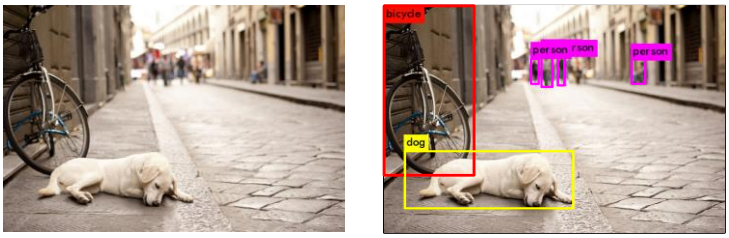
CHAPTER 5: Proposed Research Work on MAR Applications

Mobile augmented reality (MAR) applications have drawn more and more attention in recent years because of the ability to enable users to interact with virtual objects in a physical environment. With the development of the on-device neural network models and MAR developing platforms [120, 121, 70], MAR applications can be widely adopted by mobile devices in different scenarios, such as tourism, advertisement, and entertainment [122].

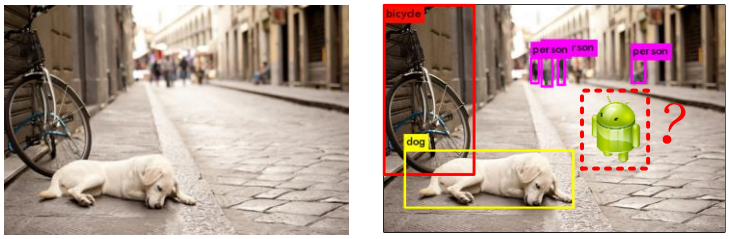
5.1 Problem Description

One of the most important and fundamental functions of MAR applications is ***object detection***. Object detection models can help to detect the category (class) and location (indicated by a bounding box) of the objects that appear in the view. The number of classes in one of the most commonly used data sets (Microsoft COCO) for training object detection models is 80 [123]. The accuracy of one of the state-of-the-art object detection models (EfficientDet) is 55.1% [124]. However, these object detection models cannot be directly implemented on mobile devices because of their limited computation resources. Thus, object detection models need to be compressed to fit mobile devices. The accuracy of the state-of-the-art on-device object detection model (MobileNet-V2) is 22.1% [124]. Both original and compressed on-device models have been widely used in MAR systems [29, 30].

In most MAR systems, mean average precision (mAP) is used as a metric for evaluating the accuracy of the adopted object detection model. However, mAP can only indicate the average performance of the object detection model. *The effectiveness of the object detection model at the frame-level is never studied.* When executing on-



(a) Original image frame (b) Correct detection with YOLOv3



(c) Detection-failure with YOLOv3-Tiny (d) Miss-detection with YOLOv3

Figure 5.1: The demonstration of detection failure and miss-detection.

device object detection in a real system, detecting nothing may occur for a large amount of image frames. These object detection executions with nothing detected are very inefficient. Object detection-failure and miss-detection are the two main reasons that lead to ineffective object detection.

With the development of model compression techniques, high complexity deep neural network (DNN) models can be compressed and deployed on mobile devices. The main side effect of model compression is that the model accuracy is decreased. As shown in Figure 5.1(c), the compressed object detection model cannot detect anything in the image frame because of the limited accuracy, which is defined as **object detection-failure**. Object detection-failure leads to poor user QoE even when the MAR system is optimized. If we adopt an object detection model with a higher accuracy (less compression) on mobile devices, for instance, Inception-ResNet [7], the inference time is 442ms per frame which is even longer than the latency of offloading the detection task to the edge server.

The number of classes most existing on-device object detection models support is less than 80. Object detection models with more classes training, such as 200 and 600 [125, 126], currently can only be executed on a server because of the high

computation complexity. For MAR applications, the objects shown in the scenes are highly likely to be beyond 80 classes. Therefore, on-device object detection models will fail to detect objects that belong to new classes beyond the 80 classes on mobile devices. The failure of detection caused by the limitation of the classes supported by the on-device object detection model is defined as **object miss-detection**. As shown in Figure 5.1(d), a virtual robot has moved into the view. However, the robot does not belong to the 80 classes supported by the on-device detection model. As a result, the object detection model fails to detect the robot. Frequent occurrences of object miss-detection also lead to poor user Quality-of-Experience (QoE) which is defined in Section 5.3.1.

5.2 Research Motivation and Challenges

Motivation. In an edge-based MAR system [24, 22, 23, 25, 29, 30, 72, 127], as shown in Figure 5.2, object detection tasks are executed on the mobile device first. If the offloading decision algorithm in the MAR system finds that offloading can bring higher user QoE, the captured frame is offloaded to the edge server. The server then executes the object detection tasks and sends back the detection results to the mobile device. The main advantage of this edge-based system is that it combines the low latency of executing object detection tasks on mobile devices and the high accuracy of the object detection model on the edge server. However, a disadvantage is that if an object detection task is offloaded to the edge server, frame offloading and result feeding back introduce additional network latency. Longer latency not only degrades user QoE, but also decreases the accuracy of the object detection results because the location of the detected objects may have changed when the mobile device receives the detection results. According to our testbed measurement results, when the frequency of offloading increases, user QoE will even degrade (more details are given in Section 5.3). In other words, offloading cannot always achieve a better QoE. That is why advanced on-device deep learning models for object detection are being

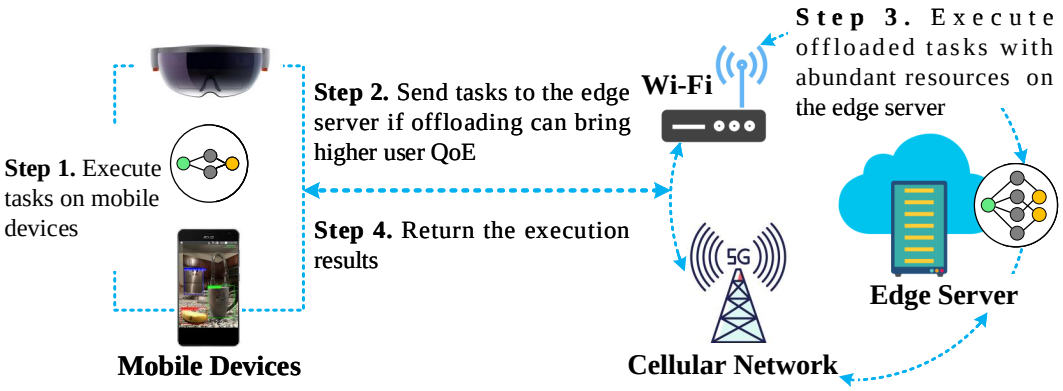


Figure 5.2: The system of edge-based MAR applications.

developed and adopted. The main problem of adopting on-device object detection models is that ineffective detection frequently occurs. In this paper, we try to address this problem from a new perspective: *reducing the occurrence of object detection-failure and miss-detection to improve the effectiveness of object detection.*

Challenges. For object detection-failure, the main challenge is that we cannot reduce the occurrence of object detection-failure on mobile devices because the accuracy of the on-device detection models is limited. Only offloading the detection tasks to the edger server can reduce the object detection-failure rate. Offloading strategies are widely used in existing MAR systems [29, 30, 31]. Existing offloading decision algorithms try to balance the accuracy and latency of executing the object detection tasks on the mobile device or on the edge server. When a detection-failure occurs, the current accuracy of the on-device object detection model becomes 0. In this case, offloading is the only option to continue executing the object detection task and to obtain a detection result. However, existing offloading decision algorithms are unaware of object detection-failure because they use the model mAP as the accuracy of the detection result for every frame, which may lead to inaccurate offloading decisions. To tackle this challenge, we propose to use frame-level detection accuracy instead of model mAP in the offloading decision algorithm.

To reduce the occurrence of object miss-detection, one approach is updating the object detection model on mobile devices. Incremental learning and fine-tuning [77, 76] can be used for updating models. Under these methods, new object classes are incrementally added to the original object detection model. The cost of adopting incremental learning is that it will take tens of minutes to complete the model updating because training the model with new data sets takes background time. Compared with incremental learning, fine-tuning takes more background time for training the new model but the new model is more robust (higher mAP). Model customization can also be used for model updating [128]. However, it focuses on using personalized data to improve the performance of the model for specific applications. To the best of our knowledge, there is no existing work on how to properly adopt a general model updating strategy in an MAR system. We are facing the following challenges to design a new model updating scheme:

(1) How to achieve fast model updating? MAR applications are mostly time-sensitive because of the interactivity. However, existing approaches such as incremental learning and fine-tuning usually take tens to hundreds of minutes background time to update a new model. Such a long updating delay significantly degrades user QoE.

(2) How to meet the diverse requirements of various MAR applications? There are many different types of MAR applications such as time-sensitive and accuracy-sensitive MAR applications. Moreover, the user movement pattern varies for different MAR applications. For each type of MAR applications, the requirements on accuracy and latency are different. It is challenging to design a model updating strategy that meets the diverse requirements of various MAR applications.

(3) How to determine when to update the model? To execute object detection with the updated new model, MAR applications need a few additional seconds to load the updated new model after completing the training into the GPU memory

of the mobile device, during which, all current object detection tasks on the mobile device are paused. Thus, model updating will add additional latency to all current object detection tasks. If updating the model whenever an on-device object miss-detection happens, the average latency of object detection will increase and user QoE will degrade. On the other hand, updating the model only after object miss-detection already happened many times will lead to frequent offloading the object detection tasks to the edge server and as explained previously, user QoE will also degrade. Therefore, determining the proper moment of model updating to balance the requirement of latency and accuracy is important and challenging.

Contributions. In this paper, we address the above research challenges and propose an improved frame-level offloading decision algorithm and an online fast AR model updating scheme to improve the effectiveness of object detection for MAR applications. Our solution focuses on reducing the occurrence of on-device object detection-failure and miss-detection. It is complementary to existing MAR offloading decision schemes [29, 30] and object tracking schemes [129, 130, 72].

5.3 Characteristic Study of MAR Applications

In this section, we show a comprehensive study on the characteristics of object miss-detection and detection-failure in MAR applications. With this study, we can obtain invaluable insights on how to design an MAR system to reduce the occurrence of object detection-failure and miss-detection and improve the effectiveness of object detection.

5.3.1 Testbed Setup

To perform the study of MAR applications, we design and implement an edge-based MAR testbed. The testbed consists of two major components: an MAR client and an edge server.

Edge Server. The edge server is developed to receive the object detection tasks

from the MAR client and send back the object detection results after executing the tasks with the object detection model on the edge server. Two main functional modules are implemented on the edge server: communication module and object detection execution module. The communication module is implemented with TCP sockect. The object detection execution module is implemented with Tensorflow. The object detection models used on the edge server are YOLOv3 and FasterRCNN-Inception-Resnet-V2 [131, 126] which can detect 80 and 600 different classes of objects, respectively. We implement the edge server on an Alienware workstation with an Nvidia RTX2020 GPU.

MAR Client. The MAR client is developed to execute the object detection tasks with an on-device object detection model first. If nothing is detected, it offloads the tasks to the edge server. To support these functions, we also implement a communication module on the MAR client which is exactly the same as the one on the edge server and an object detection execution module. We implement the MAR client on an Nvidia Jetson Tx2 with an on-device object detection model, YOLOv3-Tiny [131].

Performance Metrics. For MAR applications, three metrics are widely used to evaluate the performance:

- ***End-to-end Latency*:** The end-to-end latency L is defined as the time period from the scene capturing by the mobile device to the visualization of the result on the mobile device. It includes the execution time of the object detection model and the rendering time of the detection results. Thus, for object detection executed on mobile devices, the end-to-end latency is:

$$L_{mobile} = T^{m}_{detection} + T_{rendering}. \tag{5.1}$$

  If the object detection task is offloaded to an edge server, the end-to-end latency is:

$$L_{edge} = T_{offloading} + T^{e}_{detection} + T_{feedback} + T_{rendering}. \tag{5.2}$$

$T_{offloading}$ is the offloading time of a frame and $T_{feedback}$ is the time needed for the server to send back the detection results.

- ***Accuracy*:** The accuracy of the object detection model, A, is defined as the average IoU (intersection over union) of objects that have been correctly detected. Here, we define that an object is correctly detected if the returned bounding box has an IoU above 0.5 to the ground-truth bounding box (the same as when mAP equals 50, mAP^{50} [123]). The IoU is defined as:

$$IoU = \frac{area(Box_{gt}) \cap area(Box_{detection})}{area(Box_{gt}) \cup area(Box_{detection})}. \tag{5.3}$$

- ***User QoE*:** For MAR applications, user QoE is defined as the weighted sum of the accuracy and the end-to-end latency:

$$QoE = \mu_1 A - \mu_2 L, \quad \mu_1 + \mu_2 = 1, \tag{5.4}$$

where μ_1 and μ_2 are the weights for the accuracy and latency, respectively. For different MAR applications, the values of the weights are different. $\frac{\mu_1}{\mu_2}$ is defined as the user preference factor of an MAR application.

We measure the loading time and inference time of the object detection models adopted in our testbed and the measurement results are shown in Table 5.1. F-RCNN and Y-v3 are the abbreviation of Faster-RCNN-Inception-Resnet-v2 and YOLOv3, respectively.

Table 5.1: The performance of object detection models

Model name	Number of classes	Device type	Loading time (s)	Inference time (s)	mAP
F-RCNN	600	Server	24	0.51	55.5
Y-v3	80	Server	9.49	0.02	57.9
Y-v3	80	Jetson	12.35	0.20	57.9
Y-v3-Tiny	80	Jetson	2.99	0.09	33.1

5.3.2 Characteristics of Object Detection

To study the characteristics of detection-failure, we measure the detection-failure rate of the object detection models adopted in our testbed with two different datasets. The image dataset contains 40,000 individual image frames selected from Microsoft COCO dataset [123]. The video dataset contains randomly selected 10 video clips from the video object detection challenge [125]. The measurement results are shown in Table 5.2.

Table 5.2: The detection-failure rate of detection models

	Image Dataset	Video Dataset	Device Type
FasterRCNN	7%	6%	Server
YOLOv3	10%	7 %	Server
YOLOv3-Tiny	44%	32%	Mobile

Table 5.2 shows that adopting object detection models on mobile devices brings a large amount of object detection-failures. The accuracy of all these detection-failures is 0. When object detection-failure occurs, not only the time but also the energy consumption of the mobile device spent on executing the detection task is wasted. Table 5.2 also shows that all the object detection models can achieve a lower detection-failure rate on the video dataset compared with the image dataset. The reason is that video frames have a stronger temporal and spatial correlation compared with individual image frames, which reduces the complexity of the detection scene. Because the object detection models on the server can achieve a much lower detection-

failure rate, offloading can reduce the object detection-failure rate but also introduce additional latency.

What will happen if the on-device object detection model only detects part of the groundtruth objects on the frame? To study this, we compare the obtained detection results with the groundtruth results. There are 4 possible cases: equal, subset, superset, and overlap. For the image dataset, the distribution of these 4 cases is {0.3, 0.1, 0.4, 0.2}. For the video dataset, the distribution is {0.5, 0.1, 0.3, 0.1}. Therefore, over half of the frames have different detection results compared with groundtruth results. Thus, we should not only consider frames with a empty detection result when we design our frame-level offloading decision algorithm. ***Insight****: Existing MAR systems ignore the existence of object detection-failure. Object detection-failures significantly degrade user QoE. Thus, we should consider the frame-level effectiveness of object detection instead of considering mAP for the offloading decision algorithm.*

5.3.3 Impact of Miss-Detection on User QoE

Because there is no existing dataset or related work on object miss-detection, we have to design our own test dataset. The number of classes in the ILSVRC2015 dataset is 200. The on-device object detection model is trained with the Microsoft COCO dataset which contains 80 classes of objects. When a video snippet is chosen from the ILSVRC2015 dataset and it contains objects belonging to the 120 classes that the current on-device detection model cannot detect, object miss-detection occurs. To investigate the impact of miss-detection on user QoE, we manually control the percentage of object miss-detection by choosing different combinations of video snippets. For instance, if we want to increase the percentage of miss-detection, more video snippets that contain the objects the current on-device detection model cannot detect will be chosen.

To investigate the impact of miss-detection on user QoE, we adjust the frequency of offloading and calculate user QoE under different network data rates. Note that,

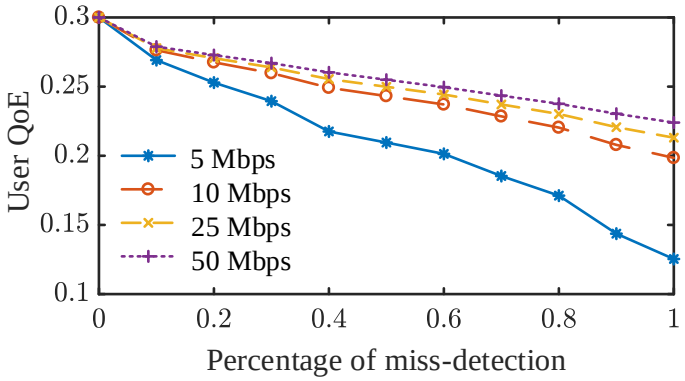


Figure 5.3: The impact of miss-detection on user QoE.

all the offloadings are caused by the on-device object miss-detection, thus, the frequency of offloading is the same as the percentage of object miss-detection. The user preference factor $\frac{\mu_1}{\mu_2}$ is set to 1, which means that latency and accuracy have equal weights for user QoE. As shown in Figure 5.3, user QoE keeps decreasing with the increasing of the miss-detection frequency. In this figure, 100% of object miss-detection means that all the object detection tasks are offloaded to the edge server, while 0% of object miss-detection means that all the object detection tasks are executed on the MAR client. With the increase of the communication data rate, user QoE drops slower. However, even with 50Mbps, user QoE still decreases by 25% when object miss-detection changes from 0% to 100%. The impact of detection-failure on user QoE is the same as miss-detection if all the detection-failure frames are offloaded to the edge server, because it does not matter whether offloading is caused by miss-detection or detection-failure.

***Insight**: Offloading cannot always help to solve the object miss-detection problem and improve user QoE. When latency and accuracy are equally important to user QoE, once the object detection task is offloaded to the edge server, user QoE will decrease. Therefore, we should reduce the frequency of offloading, i.e., reduce the occurrence of object miss-detection.*

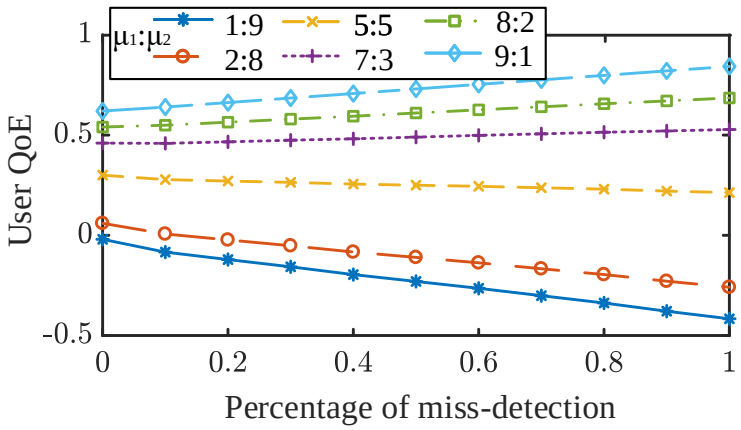


Figure 5.4: The impact of user preference on user QoE.

5.3.4 Impact of User Preference on User QoE

To evaluate the impact of user preference on user QoE, we adjust the percentage of miss-detection and calculate user QoE with different user preference factor values. The results are shown in Figure 5.4. Here, the network data rate is 25Mbps. As shown in the figure, for latency-sensitive applications (i.e., the user preference factor $\frac{\mu_1}{\mu_2} \leq 1$), offloading will degrade user QoE. On the other hand, for accuracy-sensitive applications ($\frac{\mu_1}{\mu_2} \geq 7:3$), offloading can improve user QoE.

***Insight**: The result that offloading can improve user QoE for accuracy-sensitive applications is based on the fact that the only way to obtain a higher accuracy is to offload the task to the edge server when object miss-detection occurs. However, if we can update the object detection model on the mobile device, we may obtain a similar accuracy but with a lower latency. In this way, we can also improve user QoE.*

5.3.5 Impact of User Movement on User QoE

For MAR applications, accuracy is defined as the IoU of the returned detected bounding box and the ground-truth bounding box. If object detection tasks are offloaded to the edge server, additional transmission latency will be involved compared with executing the tasks on the mobile device. The higher the latency, the higher the probability that the location of the detected object changes. If the location of

the object changes, the correct detection result may turn out to be a wrong result, because the detected bounding box is at the old location of the object, while the object has already moved to a new location in the current view. The object detection accuracy loss caused by user movement is defined as IoU_{loss}.

To study the impact of user movement on user QoE, a straightforward method is to build a model on the relationship between IoU_{loss} and end-to-end detection latency. However, this method has a problem: a higher network latency does not necessarily bring more IoU_{loss}. For instance, if a user is static, no matter how long the end-to-end latency is, there is no IoU_{loss}. The relationship between latency and IoU_{loss} is also unpredictable when the user is moving. For instance, the object may move back to the original location after a few seconds. In this case, a higher end-to-end latency brings less IoU_{loss}. However, if we can learn the user movement pattern, the relationship between IoU_{loss} and end-to-end latency latency may be found.

Object movements cause scene changes, and IoU_{loss} is caused by object movement. Therefore, we can predict IoU_{loss} based on the scene change. The temporal perceptual Information (TI) is a commonly used metric for quantifying the temporal complexity of the scenes (motions in the scene) [132, 133]. In this case, we propose a temporal complexity based regression model to quantify the impact of user movement on IoU_{loss}. TI is defined as:

$$TI = std[f(x) - f(y)], \tag{5.5}$$

where $f(x)$ and $f(y)$ represent the matrix of the pixel values of $frame_x$ and $frame_y$, respectively. *std* is the standard deviation. More motions in the scene will result in higher values of TI and also lead to a higher IoU_{loss}.

Because of lacking theoretical analysis of user movement in MAR applications, we build an analytical model based on measurement results and regression analysis. Note that regression-based modeling is one of the most widely used approaches in predicting user movement [134, 135]. 20 video snippets are randomly chosen from

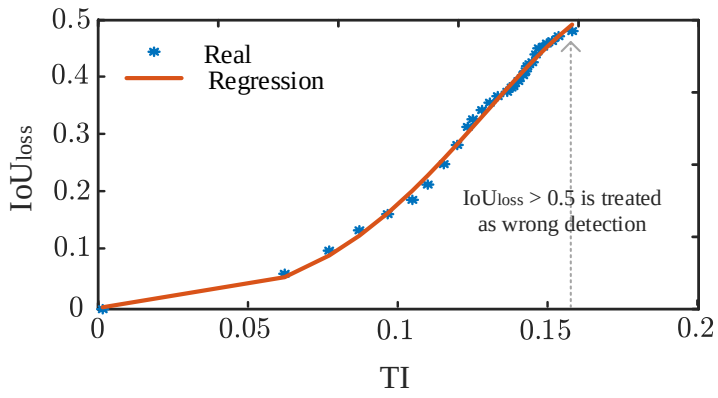


Figure 5.5: The impact of user movement on user QoE.

the video object detection challenge [125] to discover the relationship between TI and IoU_{loss}. The results are shown in Figure 5.5. We can observe that a higher value of TI results in higher IoU_{loss} (lower detection accuracy). If TI equals 0, the current scene is static and nothing is changed in the scene. As a result, IoU_{loss} equals 0. Based on our measurement results, the relationship between TI and IoU_{loss} is approximately exponential. A Gaussian regression model is adopted to learn a function between TI and IoU loss:

$$IoU_{loss} = 0.56 \cdot e^{-(\frac{TI-0.18}{0.08})^2}. \tag{5.6}$$

The root mean square error (RMSE) is applied for calculating the average model-prediction error [136], which is 0.0085 for our regression model. As mentioned earlier, if the IoU of the detection result is less than 0.5, the detection result is treated as a wrong detection result and the accuracy is 0.

***Insight**: It is very challenging to learn the pattern of user movement in MAR applications because the user is interacting with virtual objects in a physical environment. Based on our testbed measurement results, we use a regression-based model to learn the relationship between the temporal complexity of the scene and the accuracy loss of the detection result caused by user movement. As a result, when we design an*

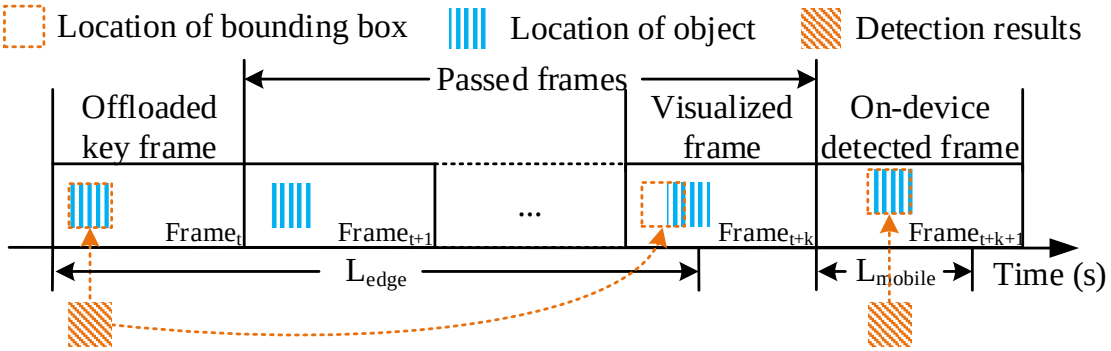


Figure 5.6: The pipeline of a real MAR system.

MAR system, we can quantify the impact of user movement.

5.4 Improved Frame-level Offloading Decision Algorithm

In this section, we formulate the offloading decision problem and detail our proposed improved frame-level offloading decision algorithm, FLOD.

5.4.1 Design Principles

As shown in Figure 5.6, in a real-world MAR system, the object detection tasks cannot be stacked and processed sequentially because the view of the user dynamically changes. For offloaded object detection tasks, the received object detection results of the offloaded $frame_t$ are visualized on $frame_{t+k}$ instead of $frame_t$. The value of k depends on the latency L^t_{edge}. Here, $frame_{t+1}$ to $frame_{t+k}$ are defined as passed frames. For on-device object detection tasks, with the support of mobile GPUs, the inference time of the on-device object detection model can be shorter than the average frame interval. For instance, the minimum requirement of real-time MAR applications is 24FPS. Then, the frame interval is about 40ms. However, the inference time of the on-device object detection model (MobileNet) is 9ms[1]. Therefore, the object detection task can be completed on-device before capturing the next frame, and there is no passed frame.

In our proposed MAR system, the offloaded frames are defined as key frames. The

[1]The inference time in all related work is obtained based on CPU performance.

next frame of a key frame has to be detected on the mobile device to evaluate the effectiveness of the on-device detection model, and it will not be fed into the object detection optimizer, because the on-device detection of this frame and the detection result evaluation occupy one frame interval. The rest of the passed frames are fed into the object detection optimizer.

In this work, the detection result from the edge server is defined as the groundtruth detection result because this is the most accurate result we can achieve. All the key frames are offloaded to the edge server in our proposed frame-level offloading decision algorithm. The detection results of these key frames are treated as groundtruth detection results to evaluate the accuracy of the detection results from the on-device detection model.

5.4.2 Problem Formulation

The objective of the proposed frame-level offloading decision algorithm is to improve user QoE through reducing the occurrence of on-device object detection-failure.

The first frame $frame_0$ is set as the key frame $frame^K$ and offloaded to the edge server to get accurate detection results. For each new offloaded $frame_t$, it is marked as the new key frame $frame^K$. After receiving the object detection results Box_t of $frame_t$, the end-to-end latency L^t_{edge}, and IoU^t_{loss} are calculated. The next frame of the key frame, $frame_{t+1}$, is always detected with the on-device object detection model. Box_{t+1} is also obtained. The detection result similarity index S^{t+1}_{box} is defined as:

$$S^{t+1}_{box} = \frac{area(Box_t) \cap area(Box_{t+1})}{area(Box_t) \cup area(Box_{t+1})}. \tag{5.7}$$

For two neighboring frames $frame_{t+1}$ and $frame_{t+2}$, the frame similarity index S^{t+2}_{frame} is used to decide whether a new key frame needs to be set or not. If S^{t+2}_{frame} is smaller than a threshold Th_f, these two neighboring frames are in different scenes. As a result, $frame_{t+2}$ needs to be set as the new key frame and offloaded to the edge server. The frame similarity index is defined as:

Algorithm 1: The improved frame-level offloading decision (FLOD) algorithm

Input: $frame_{t-1}, frame_t, \mu_1, \mu_2, Th_f, height, width$
Output: The object detection decision
while *True* **do**
  if $frame_{t-1}$ *is the key frame* **then**
    Detect $frame_t$ with the on-device detection model ;
    Calculate S^t_{box}, S^{t+1}_{frame}, QoE^{t+1}_{mobile}, QoE^{t+1}_{edge} ;
    if $S^{t+1}_{frame} > Th_f$ **then**
      if $QoE^{t+1}_{mobile} > QoE^{t+1}_{edge}$ **then**
        Detect $frame_{t+1}$ on the mobile device ;
      else
        Offload $frame_{t+1}$ to the edge server ;
        Set $frame_{t+1}$ as a new key frame ;
    else
      Offload $frame_{t+1}$ to the edge server ;
      Set $frame_{t+1}$ as a new key frame ;
  else
    if $S^t_{frame} > Th_f$ **then**
      Calculate QoE^t_{mobile},QoE^t_{edge} ;
      if $QoE^t_{mobile} > QoE^t_{edge}$ **then**
        Detect $frame_t$ on mobile device ;
      else
        Offload $frame_t$ to the edge server ;
        Set $frame_t$ as a new key frame ;
    else
      Offload $frame_t$ to the edge server ;
      Set $frame_t$ as a new key frame ;
  t = t+1;

$$S^{t+2}_{frame} = \frac{Pixel(frame_{t+2}) \cap Pixel(frame^K)}{height \cdot width}. \tag{5.8}$$

The *height* and *width* are the resolution dimensions of frames, which are the same for all the frames. If S^{t+2}_{frame} is larger than the threshold Th_f, the user QoE of detecting $frame_{t+2}$ on the mobile device or offloading to the edge server is calculated as following:

$$QoE^{t+2}_{mobile} = \mu_1 S^{t+1}_{box} \cdot S^{t+2}_{frame} - \mu_2 L_{mobile}, \tag{5.9}$$

$$QoE^{t+2}_{edge} = \mu_1 (1 - IOU^t_{loss}) - \mu_2 L^t_{edge}. \tag{5.10}$$

To achieve proposed frame-level offloading decision, $S^{t+1}_{box} \cdot S^{t+2}_{frame}$ is used to replace the mAP of the on-device detection model A_{mobile}, and the accuracy of the model

on the server A_{edge} is set to 1. If QoE_{edge}^{t+2} is larger than QoE_{mobile}^{t+2}, $frame_{t+2}$ will be offloaded to the edge server. Otherwise, $frame_{t+2}$ will be detected on the mobile device.

Based on the above formulation, we design an improved frame-level offloading decision (FLOD) algorithm, as shown in Algorithm 1. FLOD is online processed on the MAR client.

5.4.3 Frame Similarity Index

Video frames have strong temporal and spatial correlations between neighboring frames. Thus, we can detect whether the scene of the MAR applications has changed or not based on the frame similarity index. The scene has not changed if neighboring frames have a high frame similarity index. Unlike the TI introduced in Section. 5.3.5, both temporal and spatial information is needed for the frame similarity index. Frame similarity detectors (the same as frame change detectors) are widely used in object tracking systems [22, 23, 25, 130, 72]. The effectiveness of a tracking system is heavily affected by the accuracy of the detector. Thus, high complexity algorithm (e.g., random forest, support vector machine, and neural network) based detectors are popular choices. However, there is no tracking module in our proposed system. The performance of the object detection models remains the same even if we fail to detect the frame change. The frame similarity index is designed to detect whether there are significant changes in the user view. Since scene change detection is one of the most fundamental functions of video encoding, we adopt a widely used lightweight scene change detection method in the video encoding algorithm [47].

In (5.9), the frame similarity and detection similarity of the last frame are used to predict the accuracy of the on-device detection results of the next frame. According to the measurement results, 90% of neighboring frames share the same object detection results if frame similarity index threshold Th_f is set to 0.9. Thus, we can use frame similarity to predict the accuracy of the object detection result of the next frame.

5.5 Proposed Fast Model Updating Scheme

In this section, we describe the design of the proposed model updating scheme, FARMUS. Then, we formulate the model updating decision problem and propose a model updating algorithm to decide when to update the model.

5.5.1 The Design of the Proposed Model Updating Scheme

To design the proposed model updating scheme, we consider the following questions:

How to identify the occurrence of miss-detection? Without the manual control of the user, it is difficult to identify the occurrences of miss-detection on the MAR client. Thus, this function has to be implemented on the edge server. The offloaded frames are detected on the edge server using the detection model that can detect more classes. Miss-detection can be identified if a new object is found in the detection results.

How to determine when to execute model updating and meet the diverse requirements of different MAR applications? The cost of model updating is the model reloading time. During model reloading, all the object detection tasks are paused, which brings user QoE loss. The benefit of model updating is the occurrence decrease of miss-detection, which brings gains in user QoE. *The model should be updated when the gain from the model updating is larger than the loss.* Thus, we need to quantify the gain and loss caused by model updating. At the same time, we should consider the impact of user preferences and user movement patterns on user QoE. We formulate the model updating decision problem to determine the proper time for model updating and take user preference and movement pattern into consideration (detailed in Section 5.5.2).

How to achieve fast model updating? Model updating is for reducing the occurrence of object miss-detection. Thus, the updated model needs to be able to detect the classes of objects that cause object miss-detection. However, training such

a model normally takes tens to hundreds of minutes. Although the training process is executed in the background and the object detection tasks can still be offloaded to the edge server during the training time, user QoE will be degraded. To overcome this challenge, we propose to build a model pool that contains pre-trained object detection models. In this case, fast model updating can be achieved.

The next question is how many models should be prepared in the model pool? If there are k classes that cause object miss-detection, the new updated model should include all k classes. Note that, k can be calculated based on the statistic information of the offloaded object detection tasks. Assume that there are total N classes of objects and the number of classes that the on-device object detection model can detect is M. In addition, considering the usage of MAR applications in real-world, all object detection models should include m most popular classes. We can calculate the number of models needed to prepare:

$$Num_{model} = \frac{C_{N-m}^{k}}{C_{M-m}^{k}}, \tag{5.11}$$

where C represents combination. If we set N to 200, M to 80, and m to 20 (these values are selected based on the most popular object detection datasets), for the value of k from 1 to 6, the number of models needed to prepare is {3, 10, 28, 87, 273, 868}, respectively. For example, if 28 models are prepared in the model pool, for any $k \leq 3$, an object detection model can be obtained in the model pool that matches the requirement for model updating. For the case that $k > 3$, the closest matched model will be adopted for model updating. If no model in the model pool can match any class within the k classes, the incremental learning algorithm will be adopted to train a new model.

Note that, if a model in the model pool is matched, fast model updating can be achieved because the only delay for model updating is the downloading time of the model to the mobile device and the loading time of the new model to the mobile device's GPU. Even if no model can be matched, our proposed scheme will not cause

any user QoE degradation because the MAR client can still obtain the results of object detection through offloading as before.

5.5.2 Problem Formulation

The objective of the proposed model updating scheme (FARMUS) is to improve user QoE through reducing the occurrence of on-device object miss-detection.

Assume in an MAR client, the number of object detection tasks per unit time is Num_t and the percentage of object miss-detection is p. Thus, the average number of tasks executed on the mobile device and offloaded to the edge server is $Num_m = (1-p) \cdot Num_t$ and $Num_e = p \cdot Num_t$, respectively. The average user QoE per unit time is:

$$QoE_{average} = \sum_{i=1}^{Num_m} QoE_m(i) + \sum_{i=1}^{Num_e} QoE_e(i), \tag{5.12}$$

$$QoE_m = \mu_1 A_{mobile} - \mu_2 L_{mobile}, \tag{5.13}$$

$$QoE_e = \mu_1 A_{edge} - \mu_2 L_{edge}. \tag{5.14}$$

The accuracy of executing the tasks on the edge server will be influenced by the user movement pattern. Thus,

$$QoE_e = \mu_1 A_{edge}(1 - IoU_{loss}) - \mu_2 L_{edge}. \tag{5.15}$$

Denote the model reloading time on the mobile device as t_r, during which all the object detection tasks are paused. The cost of model updating is:

$$QoE_{loss} = t_r \cdot QoE_{average}. \tag{5.16}$$

For each object detection model and a particular mobile device, the model reloading time t_r is a fixed value.

After model updating, the percentage of object miss-detection is changed to $\hat{p}$, and the average number of tasks executed on the mobile device and offloaded to the edge

Algorithm 2: The proposed model updating (FARMUS) algorithm

Input: $task_n, Num_t, \mu_1, \mu_2, k, p, t_r, t_u$
Output: The model updating decision
while *True* **do**
  Receive the offloaded $task_n$
  Execute $task_n$ and send back results
  Calculate the TI between $task_n$ and $task_{n-1}$
  Collect k and p
  Search the model that contains the specific k classes in the model pool
  if *Model found* **then**
    Update $\hat{p}$
    Calculate QoE_{gain} and QoE_{loss} using (5.18) and (5.16)
    if $QoE_{gain} > QoE_{loss}$ **then**
      Send the matched model to the MAR client
  else
    Train a new mode
    Send the new model to the MAR client
  n = n+1

server is changed to $Num_{\hat{m}} = (1 - \hat{p}) \cdot Num_t$ and $Num_{\hat{e}} = \hat{p} \cdot Num_t$, accordingly. Thus, the average gain of model updating per unit time is:

$$\overline{QoE_{gain}} = \sum_{i=1}^{Num_{\hat{m}}} QoE_{\hat{m}}(i) + \sum_{i=1}^{Num_{\hat{e}}} QoE_e(i) - QoE_{average}. \tag{5.17}$$

Unlike the model reloading time, the time of model updating t_u is a relatively long time period. To fairly compare QoE_{gain} with QoE_{loss}, we set t_u to the same value of t_r. Then,

$$QoE_{gain} = t_u \cdot \overline{QoE_{gain}}. \tag{5.18}$$

Note that, the value of $\hat{p}$ depends on the result of the model matching:

$$\hat{p} = 1 - \frac{Class_k \cap Class_{model}}{k}, \tag{5.19}$$

where $Class_k$ represents the k classes of objects that cause object miss-detection and $Class_{model}$ is defined as the class set that the updated model can support to detect. Only when the updated model includes all the classes of objects which cause object miss-detection, the percentage of object miss-detection $\hat{p}$ will decrease to 0.

Based on the above formulation, we design a model updating algorithm, FARMUS,

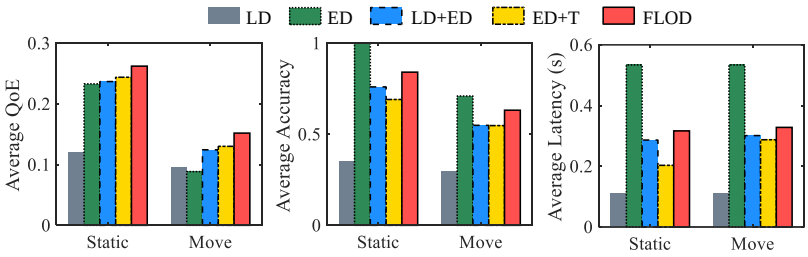


Figure 5.7: QoE analysis of FLOD.

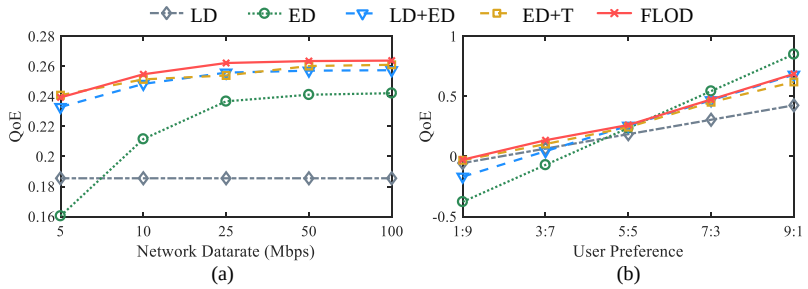


Figure 5.8: The impact of network datarate and user preference on FLOD.

as shown in Algorithm 2. FARMUS is online processed on the edge server. For each offloaded object detection task, FARMUS will decide whether to update the object detection model for the MAR client or not.

5.5.3 Performance Evaluation

5.5.3.1 Experiment Setup

MAR System. To emulate the usage of MAR applications, we use continuous video frames to simulate the sequence of object detection tasks. The original resolution of the video frames is 1280×720. During network transmission, the frame resolution is resized to 640×480. The average data size of each frame is 104 KB. The network datarate and the user preference factor are set to 25Mbps and 5:5, respectively.

Test Dataset. 50 video snippets are selected from the video object detection challenge (ILSVRC2015) [125] to build the test dataset. There are two different modes of detecting these video snippets. **Static**: the MAR client starts to detect a new frame after visualizing the result of the last frame, which means that the frame

used for detection and the frame used for visualization are the same; **Move**: the frame on the MAR client changes with a 24 FPS speed to simulate a moving scene, which means the detection result is visualized in a later frame if the detection latency is larger than the frame interval.

5.5.3.2 Baseline Algorithms

We compare our proposed two algorithms with 4 other algorithms summarized as follows:

- **LD (Local Detection):** All the detection tasks are executed on the mobile device with the on-device detection model.
- **ED (Edge Detection):** All the detection tasks are offloaded to the edge server.
- **LD+ED (baseline):** Execute the detection task on the mobile device and offload the task to the edge server if offloading can bring higher user QoE. The mAP of detection models is used to calculate user QoE.
- **ED+T:** Use tracking algorithms on the mobile device and offload the detection task to the edge server if a tracking failure occurs (a state-of-the-art approach [72]).

5.5.3.3 QoE Analysis of FLOD

User QoE. As shown in Fig. 5.7, FLOD achieves the highest average user QoE under both the static and move modes. Except for the LD algorithm, the average QoE of all other algorithms decreases under the move mode as compared with the static mode, because the accuracy of the detection results from the edge server is decreased under the move mode.

Accuracy. The results of the detection model on the edge server are treated as the groundtruth detection result. Thus, full offloading can achieve the highest accuracy performance. Under the move mode, the IoU_{loss} causes the accuracy loss of the

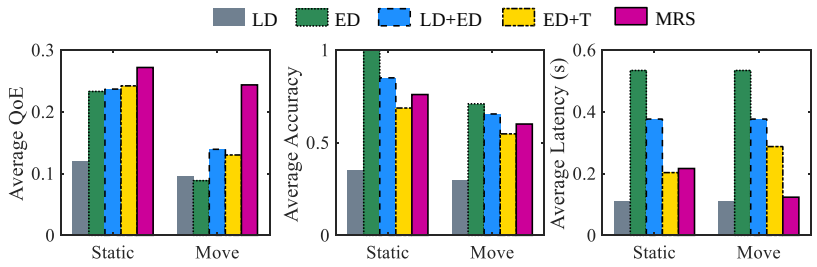


Figure 5.9: QoE analysis of MRS.

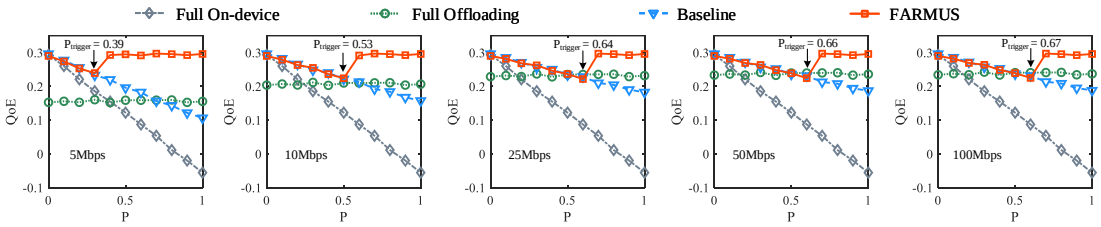


Figure 5.10: The impact of network datarate on the performance of MRS (P is the percentage of miss-detection).

detection results from the edge server. As a result, the accuracy gain of offloading is reduced and the frequency of offloading should also be reduced to achieve higher user QoE. However, the accuracy of detection-failure and tracking failure is 0. In this case, reducing the detection-failure rate by offloading these tasks to the edge server can achieve high user QoE. As shown in Fig. 5.7, FLOD achieves 10% and 21% higher accuracy as compared with LD+ED and ED+T because of the reduced detection-failure rate.

Latency. LD algorithm achieves the minimum average end-to-end latency because there is no offloading. However, its user QoE is the poorest because of the high object detection-failure rate of the on-device detection model. ED+T algorithm has a lower latency as compared with LD+ED and FLOD, because the speed of the tracking algorithm is much faster than the object detection model. LD+ED algorithm also has a lower latency as compared with FLOD because it ignores detection-failure and has fewer offloading tasks.

The impact of network datarate. We compare the performance of FLOD with other algorithms under different network datarates (user preference factor is 5:5). As

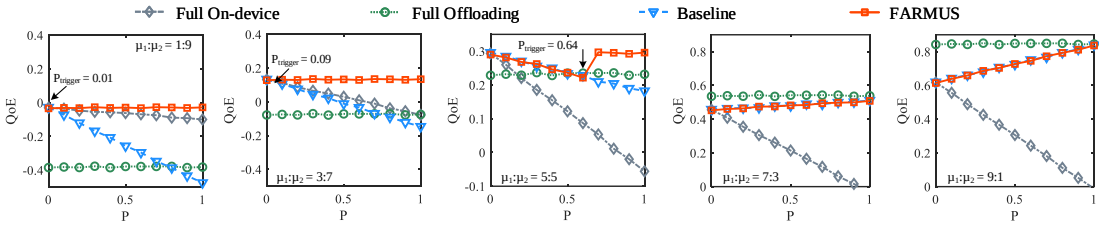


Figure 5.11: The impact of user preference on the performance of MRS.

shown in Fig. 5.8(a), FLOD can always achieve the highest user QoE under all the datarates. With the increase of the network datarate, the user QoE keeps increasing. Higher datarate results in a lower network transmission latency and a lower end-to-end latency. The network transmission latency ranges from 8 ms to 170 ms under different datarates. However, the inference time of the powerful object detection model (FasterRCNN) on the edge server is close to 500 ms. Thus, the contribution of the low transmission latency to user QoE is limited. This is why when the datarate is doubled from 50Mbps to 100Mbps, there is very limited QoE improvement.

The impact of user preference. Fig. 5.8(b) presents the user QoE under different user preference ($\frac{\mu_1}{\mu_2}$) settings (network datarate is 25Mbps). The smaller the user preference factor, the more weight the latency is counted in user QoE. Full offloading achieves the highest user QoE when $\frac{\mu_1}{\mu_2} \geq 7:3$, because more offloading results in higher user QoE for accuracy-sensitive applications.

5.5.3.4 QoE Analysis of MRS

Model replacement is triggered when the percentage of miss-detection p increases to a certain level. The p that triggers model replacement is defined as $p_{trigger}$.

User QoE. As shown in Fig. 5.9, MRS can achieve the highest average user QoE under both the static mode and move mode. Under the move mode, the miss-detection rate can be significantly reduced after the model replacement. However, all of the other algorithms have to offload the task to the edge server. As a result, MRS can reduce a large amount of offloading latency and achieve 74% improvement on

the user QoE. Compared with the static mode, $p_{trigger}$ changes from 64% to 12% under the move mode. The reason is that the benefit from the high accuracy of the powerful object detection model on the edge server decreases when user movement is considered. Thus, model replacement happens earlier as compared with the static mode.

The impact of network datarate. We evaluate MRS under different network datarates (user preference factor is 5:5). As shown in Fig. 5.10, with the increase of the network datarate from 5Mbps to 100Mbps, $p_{trigger}$ is increased from 39% to 67%. When the network datarate is doubled from 50Mbps to 100Mbps, the increase of $p_{trigger}$ is only 1%, because the change on the transmission latency contributes very limited to user QoE under these datarates. MRS can always achieve the highest user QoE for all the datarates.

The impact of user preference. Fig. 5.11 presents user QoE under different user preference settings. With the increase of the user preference factor from 1:9 to 5:5, the percentage of miss-detection that triggers model replacement $p_{trigger}$ increases from 1% to 64%. The main advantage of model replacement is to reduce the frequency of offloading caused by on-device object miss-detection. Latency-sensitive applications (with a small value of user preference factor) can gain more from model replacement when the network transmission latency is large. For instance, if the user preference factor is 1:9, the model replacement decision is made when there is only 1% of object miss-detection. After the model replacement, the percentage of offloading will be largely decreased which brings higher user QoE. On the contrary, for accuracy-sensitive applications (the user preference factor is 9:1), model replacement cannot significantly improve the user QoE. The full offloading method can achieve the best QoE. The reason is that most of the gain brought by model replacement is on the latency. If there is only a limited contribution to user QoE from the latency, the model replacement decision will not be made. As shown in Fig. 5.11(d) and (e), MRS works

exactly the same as the LD+ED algorithm, which means that model replacement is not performed.

5.6 Proposed Smart-Decision Algorithm

With the development of the hardware and software platforms, we can implement the deep learning model on the mobile device for mobile augmented reality (AR) applications. However, not all mobile AR tasks can be finished on mobile devices. Meanwhile, the limited computation resources on mobile devices are still the main obstacle to achieve realtime mobile AR applications. To tackle these challenges, we proposed a smart-decision framework that combines the advantages of the on-device mobile AR system and the edge-based mobile AR system to achieve real-time object recognition. High computation complexity tasks will be offloaded to the edge servers. Low complexity tasks will be executed on mobile devices or the edge server depending on the network latency. To overcome the dynamic changes of the network condition and the limitations of the on-device deep learning models, we design a cache and matching algorithm on mobile devices to enhance the performance of the recognition tasks.

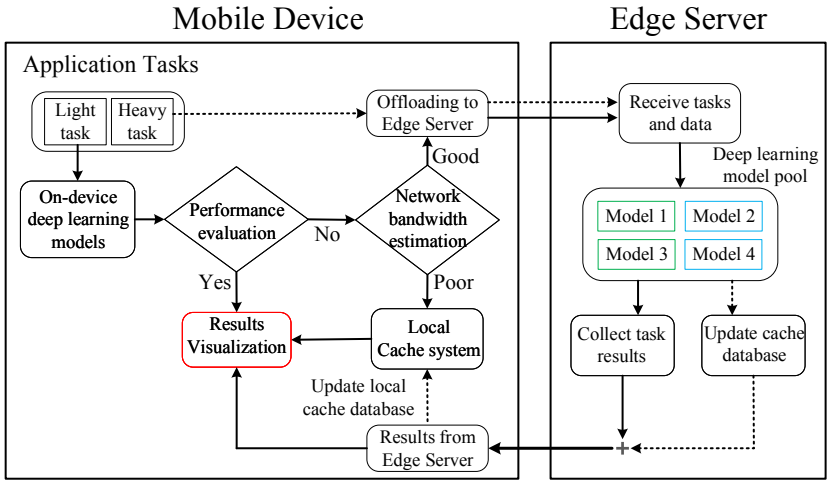


Figure 5.12: The system framework.

5.6.1 System Architecture

In this section, we present our proposed system in detail. As shown in Figure 6.2, our system is composed of two parts: mobile device and edge server. We divide all the recognition tasks into two categories. A task that can be executed on mobile devices are defined as the light task T_L, e.g., classification and detection. A task that has to be offloaded to edge servers is defined as the heavy task T_H, such as the semantic analysis. The smart decision algorithm is shown in Algorithm. 3.

Algorithm 3: The smart decision algorithm

Input : A set of tasks $\{T_1, T_2, ..., T_n\}$, real-time network throughput $\mathcal{B}$, network bandwidth threshold N_σ;
Output: The decision of where to execute the tasks;

```
Initialize the task index i = 1;
while i <= n do
    if T_i ∈ T_H then
        Offload the task T_i to edge server;
    else
        Execute the task T_i with the on-device deep learning model;
        Receive performance feedback P;
        if P = Poor then
            if B > N_σ then
                Offload the task T_i to edge server;
            else
                Execute the task T_i with the cache system;
    i=i+1
```

5.6.1.1 Mobile device

For each mobile AR application, the mobile device will keep capturing the video frames using the camera. For each video frame, there will be different recognition tasks needed to be executed. The heavy task will be directly offloaded to the edge servers because mobile devices cannot provide enough computation resources. The light tasks will be executed locally with the on-device deep learning models. However, since the performance of the on-device deep learning models is not robust as compared with the deep learning models on the servers. In this case, we design a

performance evaluation module. If users are not satisfied with the performance of current recognition tasks, the next task will not be executed with the on-device deep learning models. First, the network bandwidth will be estimated. If the network bandwidth is higher than a threshold T_b, the light task will be sent to the edge server. However, if the network bandwidth is lower than T_b, it means that the user is under the poor network condition. In this case, if we still do the task offloading, the performance will be even worse. The reason is that the mobile AR application is very sensitive to the latency. The scene will change a lot when you get the results from the edge server. To overcome this issue. We design a cache and matching algorithm on the mobile device, which can help to improve the performance of light tasks on the device with a lower latency.

In the cache and matching module, we extract the image features using the feature extraction methods such as SIFT. These features are used to match with the objects stored in the cache on mobile devices. After a successful matching object is found, the matching process is ended, and the detection result is presented to the user. There are many different image feature extraction algorithms, the time cost and result quality of applying these algorithms are different. In addition, the size of the cache on the mobile is also vital for the matching system. If the number of objects in the cache is very large, it will take a long time to find a successful match, and it will neutralize the advantages of the cache system. In this case, we build a hierarchical cache system. For each video frame, we match the frame with high-ranked objects for each class. And once the users offload the tasks to the edge server, the cache system will be updated. There are several trade-offs between the time and the performance in cache and matching algorithms. We show more details in Section 5.6.2.

5.6.1.2 Edge server

On the edge server, we will receive the data and tasks from the users. For different tasks, we have different algorithms and models implemented on the edge server. Edge

server can provide the service for both heavy and light tasks. For heavy tasks, the edge server will execute the tasks and send back the results. At the same time, the edge server will backup all the recognition results and extract the image features of the objects. These results and image features will be used to update the cache database. For the light tasks, if we get poor performance with the on-device deep learning model, or we cannot get a successful match with the cache and matching module, the image frame will be sent to the edge server. Besides sending back the results of light tasks to the mobile device, we will also synchronize the cache on the mobile device with the database on the edge server. In this case, we can still achieve good performance by using the cache and matching module when the network bandwidth is low. Another reason for building the database on the edge server is that we can share this database with multiple users.

5.6.2 Cache and Matching Module

As shown in Figure 5.13, we propose a feature matching method for the classification and object detection tasks. One of the main challenges of mobile AR is the limited computing capacity of mobile devices. A high network latency will occur if we offload all the computation to the cloud, or the edge server. We propose a feature matching method that only needs low complexity computation on the mobile device to achieve classification and object detection tasks. Mobile devices can support enough computing resources for feature extraction and matching algorithms. The main process is as following: First, we build a cache data store to store objects and corresponding feature vector sets on mobile devices. This cache store will be updated by the edge server. Then, we extract the features of the target image. After that, we match objects in the cache data store with the target image feature set one by one. For each feature vector in the vector sets, we measure the closest distance between it and the features of the target image. If the closet distance is less than a threshold, we call it a successful match feature vector; if the number of the total

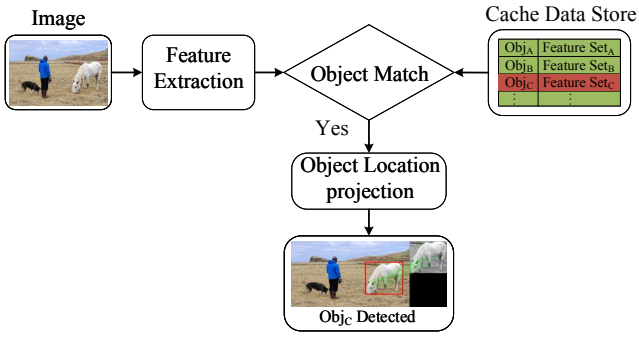


Figure 5.13: The cache matching system.

successful match feature vector is large than a threshold K, it means the object is in the target image. Then, we project the successfully matched feature vectors on the target image, and a projection box is obtained. In this paper, if the project box cover over 80% of the object, we treat it as a successful projection. In this case, we can achieve low complexity object detection.

5.6.2.1 Features extraction and matching algorithms

The image feature is like a signature of the image. For different feature extraction algorithms, the feature vector dimensions and the matching algorithms are both different. In this paper, we use three feature extraction algorithms, SIFT [64], SURF [65] and ORB [66]. As shown in Table 5.3, we compare the differences of several popular feature extraction and matching algorithms. For different feature extraction methods, the feature extraction and matching time costs are different. For instance, ORB uses the binary number to construct the feature vector and uses the Hamming method to measure the distance of the feature vectors for matching. So it is the fastest among these three methods. However, when we use the same number of feature vectors for matching, SIFT can achieve higher matching accuracy.

Table 5.3: Feature extraction and matching algorithms

Algorithm	Dimension	Data type	Time(s)	Matching method
SIFT	128	uint	0.05	FLANN
SURF	64/128	uint	0.01	FLANN
ORB	256	Binary	0.003	Hamming

Except the time cost, the number of image features extracted from each image is another important factor that we need to take into consideration. Extracting more image features means a better presentation of the image, and we can get better performance in the matching process. Meanwhile, it will take more computing time and matching time. So the number of features should be properly selected.

5.6.2.2 Cache and Object Database

On the edge server, we create a large object database. In the object database, we store about 10000 objects and the corresponding feature vector sets of each object. These objects are obtained from 20 classes of images in ImageNet dataset [137]. When we offload the tasks to the edge server, all the objects detected by the recognition models will be added into the database with a high rank value. On the mobile device side, we design a cache system, which will fetch the high ranked list of objects from the database on the edge server.

$Class_{ID}$	Object $_{UniqueID}$	Object Feature
$Class_A$	$Rank_1$	Feature Set_1
	$Rank_2$	Feature Set_2
	⋮	⋮
$Class_B$	$Rank_1$	Feature Set_1
	⋮	⋮
⋮	⋮	⋮

Figure 5.14: The hierarchical cache data base.

As shown in Figure 5.14, there are three layers in the cache database, and the first layer is the class label. The objects belong to the same class are collected together. The second layer is the unique ID of each object. In this paper, the unique ID is the rank value from the edge server. High ranked objects will have a high priority to be matched. When the number of objects in the cache data store is larger than a threshold, the objects with the lowest rank will be removed from the cache data store. The last layer is the feature set of each object. This feature set composes the

fingerprint of the object. We use these object fingerprints to match the feature set of the video frame.

As mentioned early, we match each object in the cache with the target image until we get a successful match. If we put the total database into the cache on the mobile device, the total time for matching is much higher than offloading tasks to the edge server. If there are only a few objects in the cache mobile device, the successful matching rate will be reduced.

5.6.2.3 Trade-offs

The speed and accuracy of the cache and matching system are influenced by three key parameters, the methods of feature extraction, the number of features to be extracted for each image, and the cache data store size. Considering the robustness of the system and the time cost, we choose SURF as the main image feature extraction method. As shown in Figure 5.15, with the increasing of the cache size, the time to finish the matching keeps increasing. At the same time, the matching time cost will be much higher if we set the image features size to 800 compared with other settings. To fit the real time processing requirements for the mobile AR applications. We set the cache size to 100 and the image feature size to 500.

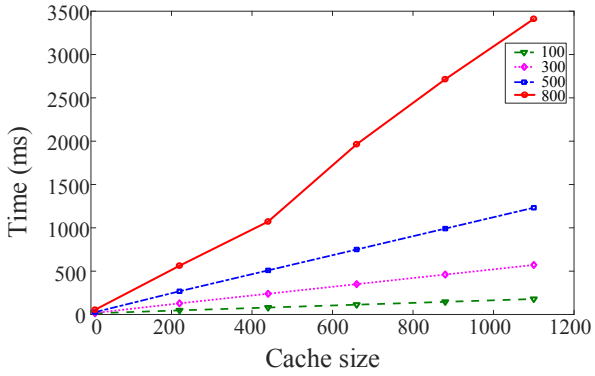


Figure 5.15: The trade-offs in the cache system.

5.6.3 Performance Evaluation

5.6.3.1 Testbed set up

To evaluate the performance of our proposed system. We build a testbed to implement all the modules proposed in our framework. We simulated a network with MiniNet combined with the SDN controller ONOS. MiniNet is used to generate the network nodes and links, while the ONOS is used to control the routing and data flows. The Mininet and the ONOS are implemented on an HP EliteDesk 800 G2 workstation.

As shown in Fig. 5.16, there are are total 30 switch-nodes$N = \{n^1, n^2, ..., n^{30}\}$, and 69 links in the network. The link data rate is 50 Mbps, and the delay of the links follows the normal distribution with $\mu = 6ms, \sigma = 1$. The blue node represents the edge server. It is bridged with the Jetson AGX Xavier to support the recognition tasks for the users. We have two users in our testbed system. Each user is an ASUS Zenfone-AR mobile phone. We use Tensorflow Lite to develop an image classification and an object detection android applications on mobile phones. Since the mobile phones cannot be directly bridged to the simulated network, we use two Jestons-Tx2 as the mobile device access node, which is represented by the 2 purple nodes in the simulated network. The mobile phones and Jetson-Tx2s are connected through a router. The mobile phone application will send the data to the Jetson-Tx2 at first. Then, the Jetson-Tx2 will send the received data to the edge server. Once Jetson-Tx2 receives the results from the edge server, it will send back the results to the mobile phone. The time cost between the mobile phone and the Jetson-Tx2 will not be included in the end-to-end latency. We send total 1500 frames from the two mobile phones to the edge server.

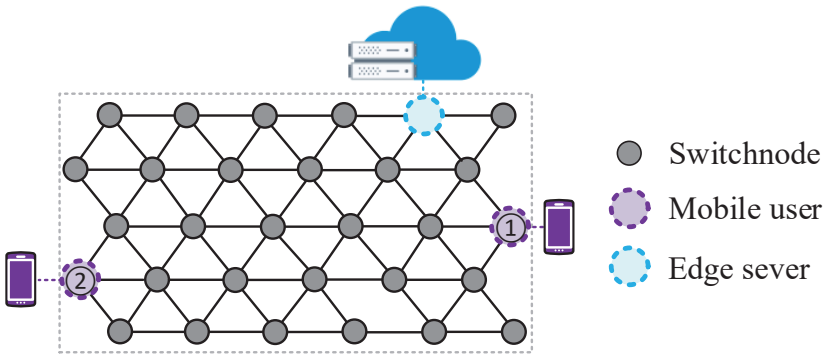


Figure 5.16: The network topology of the simulated network.

5.6.3.2 Experiment Results Analysis

In this section, we evaluate the performance of the proposed smart decision framework through the experiments on our testbed. Since the heavy tasks will be only executed on the edge server, we do not put the performance of heavy tasks in this part.

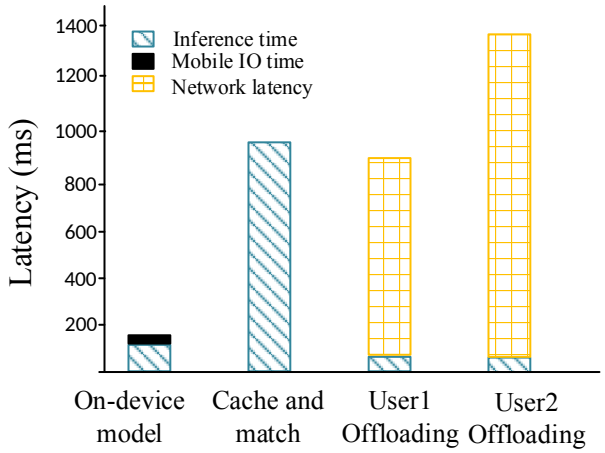


Figure 5.17: The end-to-end delay of classification application.

Inference Accuracy: We implement an image classification and an object detection algorithms on the edge server, and we develop an image classification and an object detection android applications on two Android phones. As shown in Fig. 5.17 and Fig. 5.18, the end-to-end delay for running the classification and detection applications on androids phones are 155ms and 224ms, respectively. The inference times

are 115ms and 184ms respectively. If we choose to offload the tasks to the server, the network latency is involved. Since the distances between the users and the edge server are different, the network latency for two users is different. The network latency is 850ms and 1380ms for $user_1$ and $user_2$, respectively. On the edge server, the inference times for classification and detection are 64ms and 111ms, respectively. For $user_1$, the end-to-end delays for classification and detection are 924ms and 961ms respectively. For $user_2$, the end-to-end delays for classification and detection are 1452ms and 1501ms, respectively.

Table 5.4: The mAP of different models

Model	mAP
$SSD300(server)$	81.2%
$SSD + MobileNet(on - device)$	72.7%
$Cache + matching(on - device)$	76.3%

Miss classification happens a lot for the on-device classification and detection applications. For example, an apple is mistakenly recognized as a ball. When this error happens, we can choose to do the classification detection with our proposed cache and matching system. The end-to-end delay for the on-device cache and matching system is 1039ms. If we do not need to know the location of the project, the delay is 980ms. We can find that if the network condition is good, using the cache and matching system is not necessary. The reason is that offloading the task to the edge server is faster. However, when users are under the poor network condition, like $user_2$, the cache and matching system can help users to get better accuracy with less time cost compared to offloading the tasks to the edge server.

End-to-end Latency: The inference accuracy is decided by the deep learning models. High complexity models can achieve a higher precision. On the mobile object detection applications, we use the ssd+MobileNet model. On the edge server, we use the SSD300. The mean Average Precision(mAP) of these models are listed in Table 5.4. The mAP of the cache and matching system is lower than the model on

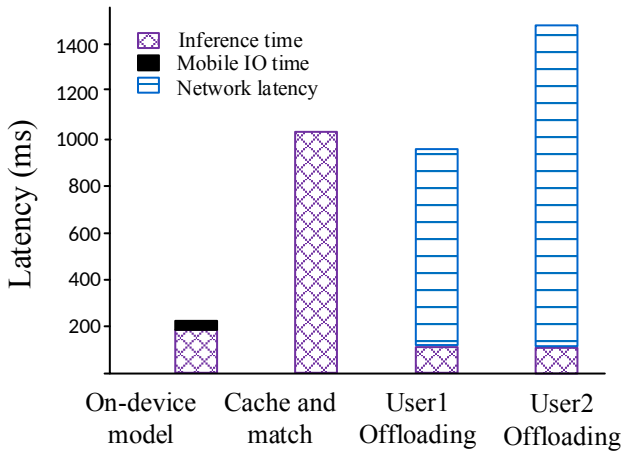


Figure 5.18: The end-to-end delay of object detection application.

the edge server. The reason is that the cache size is limited, not all the objects will be successfully matched.

CHAPTER 6: Proposed Network Partitioning for Efficient Management of UDN

In this work, we propose a wireless big data-driven virtual network function placement framework which integrates the wireless big data analysis, machine learning, and network optimization to enhance the performance and efficiency of managing ultra-dense networks [138]. The proposed framework is composed of a data processing module, an offline machine learning module and an online virtual network function placement module. The data processing module collects the mobile network records and extracts the network features. These network features are used for quantifying the relationships among BSs and partitioning the network. We investigate both BS-based and grid-based network partitioning.

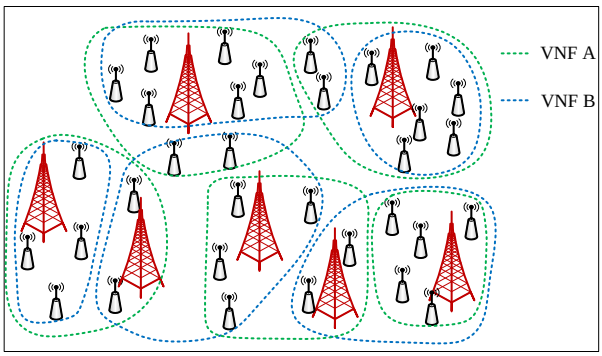


Figure 6.1: The proposed framework with two network functions.

The offline module performs the network partitioning by using the hierarchical clustering analysis (HCA) method [44]. The performance of network partitioning solutions can be evaluated by comparing the performance of network functions applied on the network partitions (corresponding to different network partitioning solutions). Moreover, we analyze the relationship between the number of sub-RANs and the performance of the network function. The analysis results are utilized to train a

prediction model to obtain the optimal number of sub-RANs under different traffic conditions. Based on the prediction results, the online module adopts the K-medoids clustering algorithm [139] to partition the network into sub-RANs and place the network function in each sub-RAN. Each sub-RAN will be managed independently by the network functions. Additionally, our framework can be applied to many network management functions, such as traffic load balancing, spectrum allocation and so on. In a wireless network, these network functions should be able to work simultaneously. Figure 6.1 shows how two network functions simultaneously manage the network. Each network function uses its own network partitioning solution.

6.1 Framework Overview

In this section, we present an overview of the framework for placing virtual network functions (VNFs) in an ultra-dense mobile network, as illustrated in Figure 6.2. To reduce the complexity of network management, the proposed framework partitions the network into sub-RANs and place the VNFs in each sub-RAN. The framework consists of a data processing module, an offline machine learning module, and an online network function placement module. We list the notations in Table 6.1.

Table 6.1: Notations

K	The optimal number of sub-RANs
n	The number of base stations
$\mathcal{B}$	The set of base stations. $\mathcal{B} = \{B_1, B_2, \cdots, B_n\}$
$\mathcal{U}$	The set of users in the network.
$\mathcal{P}$	The networking performance of network functions.
$\mathcal{V}$	The computation performance of network functions.
F	The network features. $F = \{F_d, F_t, F_u\}$
Γ	The traffic load of the network. $\Gamma = \{\gamma_1, \gamma_2, \cdots, \gamma_n\}$
T	The set of time slots in a day. $T = \{t_1, t_2, \cdots, t_{144}\}$
γ	The traffic load on a base station.
γ^G	The traffic loads on BSs within a grid.

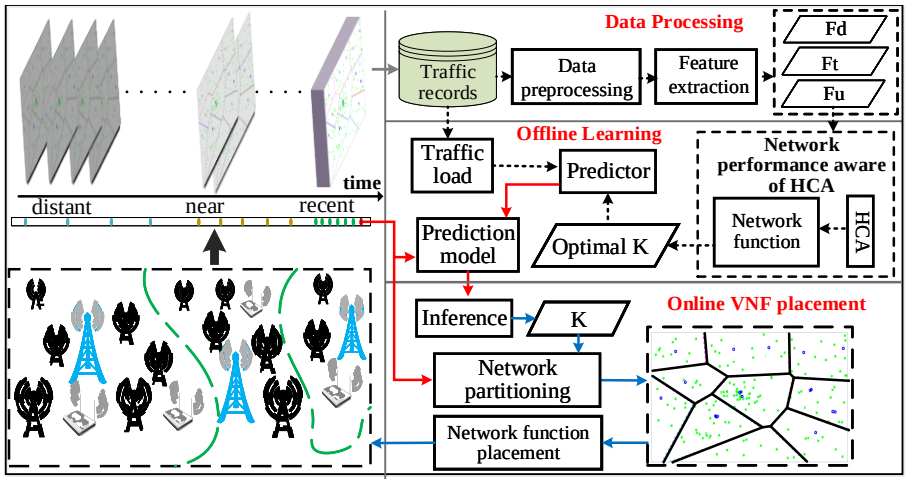


Figure 6.2: The data-driven virtual network function placement framework.

6.1.1 Data Processing

The data processing module mainly includes the mobile traffic data collection, data processing, and network feature extraction. The mobile traffic records in each BS are collected and stored in a traffic record database. These data are used for the network feature extraction and network partitioning analysis. In consideration of the redundant and conflicting information contained in the original data, we filter the traffic record data before using them to extract the network features. After having cleaned up the data, we segment the traffic records into thousands of small chunks. Each chunk contains a 10-minutes traffic record. The 10-minute segmentation is chosen because it is the smallest time interval in which a cellular tower can experience non-zero traffic [94]. These chunks of traffic records will be classified into three parts according to the time of collection: distant (one month ago), near (2-4 weeks ago) and recent (now to two weeks ago). Since the prediction model needs to train with a large amount of data, we use all recent chunks and a portion of the distant and near chunks to train the prediction model.

In this work, we partition the network for the traffic load balancing function. Three networking features are extracted from the traffic record database to quantify the relationships among the BSs. For different VNFs, the definition, number, and weight of

the network features are different. In this paper, we use Traffic Load balancing(TLB) as an example of VNF, and three network features are extracted. The first feature relates to the geo-distances among BSs. The second feature is the similarity of the traffic trends in BSs. The third feature is the number of common users in BSs during a given period of time. The extracted features are stored in the feature database and will be sent to the offline machine learning module to train for the network partitioning solutions.

6.1.2 Offline Machine Learning

The offline machine learning module consists of two components: the network performance aware clustering analysis and prediction model training. Based on extracted networking features, we define a distance function to quantify the relationships among BSs. The hierarchical clustering analysis (HCA) algorithm is adopted to partition the network. The HCA algorithm generates a clustering tree. Each layer of the clustering tree represents a network partitioning solution. The index of the layer indicates the number of BS clusters[1] in the corresponding partitioning solution. For example, the network partitioning solution obtained from the 10th layer of the clustering tree will partition the network into 10 BS clusters. The network performance of each network partitioning solution is evaluated by the network functions. After evaluating the network partitioning solutions obtained from each layer of the clustering tree, we can derive the optimal network partitioning solution. Thus, the optimal number of BS clusters can be obtained.

The clustering analysis and the performance evaluation of the network partitioning solution incur very high computational complexity. They are not appropriate for realtime network partitioning and optimization. To perform real-time network partitioning, we mine the wireless big data to learn the relationship between the number of

[1]In the paper, we define a sub-RAN as a cluster of BSs. Hence, we use the sub-RAN and BS cluster interchangeably.

sub-RANs and the performance of the network function, and then train a prediction model to predict the optimal number of sub-RANs under different traffic conditions. For different VNFs, the models obtained by offline training are different because the extracted network features used for one VNF are different from those for another VNF. Nevertheless, the training procedures are same for all VNFs. The online module will use this prediction model to predict the optimal number of sub-RANs, and perform the network partitioning in realtime. The virtual network functions will be placed on the partitioned network to manage each sub-RAN.

In this paper, we choose two different granularities to partition the network. The first one is BS-based network partitioning, in which the BS is the smallest unit for partitioning the network. We partition the network into several sub-RANs, and each sub-RAN contains multiple BSs. The second one is grid-based network partitioning, in which we divide the network into small grids. In this paper, we set the size of each grid to 100*100 m^2. After that, we partition those grids into different sub-RANs. The grid is the smallest unit in the sub-RAN. There are mainly two motivations of adopting different scales in network partitioning. The first reason is that considering a large-scale area like a city or a downtown area, there are tens of thousands of BSs, in which BS-based network partitioning may not be efficient. Another reason is when we train the prediction model, each BS is seen as one dimension of the input vectors in the prediction model. Thus, the dimension of the input vector is very high under BS-based network partitioning. This may cause a high prediction error rate. The dimension of the input vector will significantly decrease when we use the grid-based network partitioning method.

6.1.3 Online Virtual Network Function Placement

Given the offline trained prediction model, the optimal number of clusters can be derived based on the current network traffic load conditions. Then, the network will be partitioned into clusters based on low-complexity clustering algorithms such as

K-medoids [140], see Section 6.4.1 for more details. Once the network is partitioned into sub-RANs, a virtual network function is placed for each sub-RAN to optimize the network performance. The virtual network function should be the same one used for evaluating the network partitioning solution in the offline training module. In this paper, we use, as an example, a traffic load balancer [36] as the network function.

The proposed realtime VNF placement scheme consists of four steps. First, the traffic loads on each BSs in the last 10 minutes is collected. Second, the traffic loads are fed into the well-trained prediction model and the optimal number of sub-RANs, K, can be obtained. Third, with this prediction, the BSs are partitioning into K sub-RANs using K-medoids clustering algorithm. The prediction and partitioning processes only take less than 1 minute. Fourth, the virtual network function is placed for managing each sub-RAN.

6.2 Data Processing

The data processing module consists of three functions: data collection, data pre-processing and the networking feature extraction.

6.2.1 Data Collection

In our framework, mobile traffic records on each BS will be continuously collected and stored in the traffic record database. As shown in Table 6.2, a mobile traffic record is composed of six attributes including the identification number of a BS, the geographical location of the BS (latitude x and longitude y), the identification number of a user, and the start time T_s and end time T_e of the user's data connection on the BS.

Table 6.2: Traffic record samples

BS ID	**x**	**y**	User ID	T_s	T_e
10027	45.75	126.62	79xxxxx	1210120149	1210120252
13852	46.01	127.39	65xxxxx	1212151547	1212151624
16212	44.77	128.45	61xxxxx	1218124335	1218124357

6.2.2 Data Preprocessing

The data preprocessing addresses three problems in the traffic record data set. First, there is inconsistent and redundant information in the traffic records. For instance, in some records, the data connection ending time is earlier than the starting time. The data set also contains duplicated traffic records. These records are eliminated from the database. Second, there are incomplete data records. In some data records, some attributes are missing, e.g., BS ID and geo-locations. To solve this problem, we store all BS' IDs and their corresponding geo-locations in a database. Once the geo-location attribute is missing in a traffic record, we can locate the BS ID in the database and obtain the corresponding geo-location, and vice versa. Third, to preserve the privacy of the users, we anonymize the user identification numbers in the traffic records.

6.2.3 Networking Feature Extraction

In this section, we present the networking feature extraction and analysis for network partitioning. To partition the network, a distance function is required to quantify the relationships among the BSs. Therefore, we extract three networking features to define the distance function.

For BS-based network partitioning, the first feature is the geo-distances among BSs. The second feature is the similarity of the traffic trends in BSs. The third feature is the number of common users in BSs during a given period. Here, each networking feature is defined as one factor that may impact the relationships among BSs. For grid-based network partitioning, we use the same features, but these features are calculated in different ways from the BS-based network features.

These networking features are extracted from a mobile traffic data set containing 2.8 billion traffic records collected from an operating mobile network. The extracted features are stored in the feature database and will be sent to the offline machine

learning module for partitioning the network. Since the high mobility of the mobile network, we update the network features database periodically with the recently collected network data in response to the network traffic variations.

6.2.3.1 The Geo-Distance Feature: F_d

The geographical distance between two BSs is one important factor to measure the relationship between the BSs. When the BSs are close to each other, their coverage areas will be overlapped. A larger overlapped area indicates a tighter coupling since there will be more users in the overlapped area. From the data set, we can get the latitude and longitude of each BS. Then, we use the Spherical Law of Cosines formula to calculate the geo-distance between the two BSs [141]. We define x_i and y_i as the latitude and longitude of the ith BS, respectively. φ_i and λ_i are the corresponding radians of the latitude and longitude. R is the radius of the earth. The geo-distance between the ith and jth BSs is expressed as:

$$d_{ij} = \arccos(\sin\varphi_i \sin\varphi_j + \cos\varphi_i \cos\varphi_j \cos\Delta\lambda)R, \tag{6.1}$$

where $\varphi_i = \frac{x_i\pi}{180}$ and $\Delta\lambda = \frac{y_i\pi}{180} - \frac{y_j\pi}{180}$.

We assume that the transmission range of a BS is 1.5 km. The distance between a BS and one of its neighboring BSs is up to 3 km. The CDF of inter-BS distances between a BS and its neighboring BSs is derived from the mobile network data set. As shown in Figure 6.3(a), the geo-distances between BSs and their neighboring BSs are diversified. This diversity indicates that the geo-distances can be used to measure the relationships among BSs. For instance, the distances between BS 1 and 35% of its neighboring BSs are less than 1.5 km. This indicates that BS 1 may have tighter relationships with these 35% of neighboring BSs. Therefore, the geo-distance between two BSs is recognized as one of the features for BS-based network partitioning.

In grid-based network partitioning, we adopt a different method to calculate the

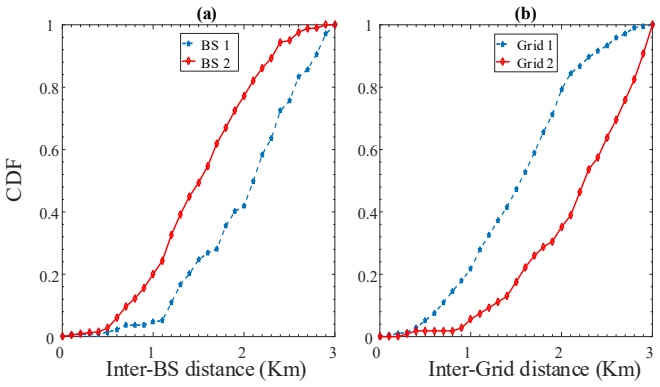


Figure 6.3: The CDF of neighbors' distances.

geographical distance feature. As shown in Figure 6.4, each grid is identify by its index (x,y). We use the Euclidean distance to measure the distance among the grids, i.e., the distance between grid G^i and G^j is:

$$d_{ij}^G = \sqrt{(G_x^i - G_x^j)^2 + (G_y^i - G_y^j)^2} \tag{6.2}$$

(n,1)	(n,2)	...	(n,m-1)	(n,m)
(n-1,1)	...	...	...	(n-1,m)
...	...	...	...	...
(2,1)	...	...	...	(2,m)
(1,1)	(1,2)	(1,3)	...	(1,m)

Figure 6.4: The grid map.

The CDF of inter-grid distance is shown in Figure 6.3(b). The distances between grids and their neighboring grids are also diversified. Therefore, the geo-distance between two grids can be recognized as one of the features for grid-based network partitioning.

6.2.3.2 The Traffic Trend Similarity Feature: F_t

The mobile traffic trend measures the mobile traffic diversity over time in a BS. If two BSs are tightly coupled, they will have similar traffic trends. Figure 6.5 presents the traffic trend analysis among three neighboring BSs. We define the duration of a time slot to be 10 minutes. Then, a day consists of 144 time slots. To derive the traffic trend, we calculate the cumulative traffic load in a BS in each time slot. Here, we assume that all users' traffic data rates are the same. Hence, the traffic load generated by a traffic record in each time slot is equal to the data rate multiplied by the duration of the data connection. Since our objective is to measure the similarity of the traffic trend, we normalize the values of all the traffic loads into the range of 0 to 1.

Denote $\gamma_{i,j}$ as the accumulative traffic load in the ith BS in the jth time slot. Then, $\Gamma_i = \{\gamma_{i,1}, \gamma_{i,2}, \cdots, \gamma_{i,144}\}$ represents the traffic load sequence in the ith BS. Figure 6.5(a) shows instantaneous traffic loads in these BSs. Traffic loads on different BSs reach the maximum values at different times. From 13:00 to 16:00, BS3 keeps a low traffic, but the traffic increases on BS1 and BS2. To analyze their traffic trend in a BS, its traffic load sequence is transformed into the frequency domain in which the low-amplitude frequencies are filtered out. The residual high-amplitude frequencies reflect the traffic trend in the BS. As shown in Figure 6.5(b), BS 1 and BS 2 have similar traffic trends and their traffic trends are notably different from that in BS3.

Denote $\gamma^G_{i,j}$ as the accumulative traffic load in the ith Grid in the jth time slot. $\gamma^G_{i,j}$ is calculated by adding the traffic loads of all BSs inside the ith Grid. $\Gamma^G_i = \{\gamma^G_{i,1}, \gamma^G_{i,2}, \cdots, \gamma^G_{i,144}\}$ represents the traffic load sequence in the ith grid. As shown in Figure 6.5(c) and 6.5(d), we can find that Grid 1 and Grid 2 have similar traffic trends, and their traffic trends are notably different from that in Grid 3.

The traffic trend similarity between BSs can be quantified by using the discrete Fréchect (DF) distance [142], which can calculate the similarity of two curves regard-

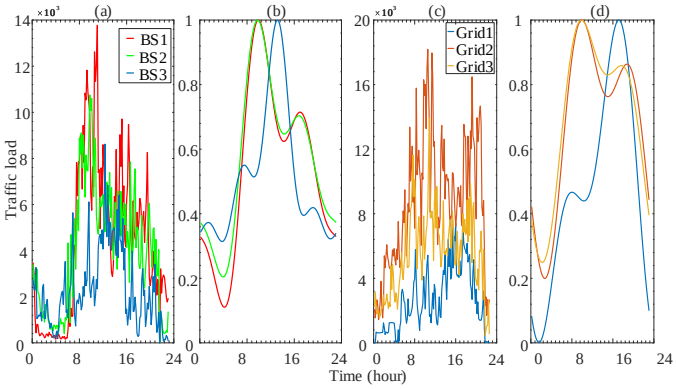


Figure 6.5: The traffic trend analysis.

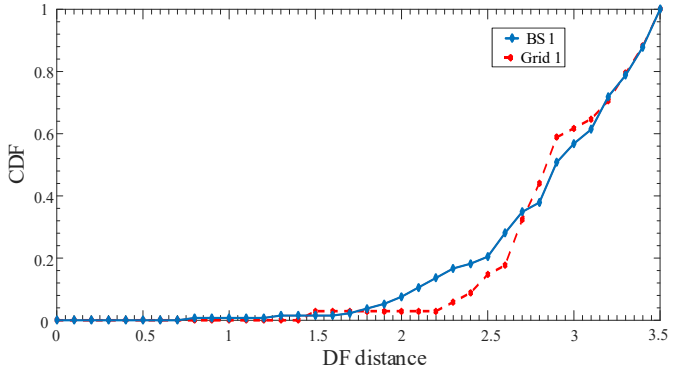


Figure 6.6: The CDF of the traffic trend distance.

less of the shift, scale, and rotation of the curves. Denote $\delta_{i,j} = FrechetDist(\Gamma_i, \Gamma_j)$ as the DF distance between the ith and jth BSs. For the traffic trend analysis shown in Figure 6.5(b), $\delta_{1,2} = 0.7$ and $\delta_{1,3} = 2.6$. It is shown that when $\delta_{i,j}$ is larger than 2, a significant difference can be identified between the traffic trends of the ith and jth BSs. We use the same method to measure the traffic trend similarity between Grids. Figure 6.6 shows the CDF of the DF distance between the traffic trend of BS 1 and that of its neighboring BSs and the CDF of the DF distance between the traffic trend of Grid 1 and that of its neighboring grids, respectively. For BS 1, about 90% of neighboring BSs have different traffic trends. The CDF indicates that the traffic trend similarities vary among BSs. This analysis advocates for using the traffic trend similarities among BSs as a feature for partitioning ultra-dense mobile networks. For Grid 1, about 96% of neighboring grids have different traffic trends. The CDF indicates that the traffic trend similarities also vary considerably among

Grids.

6.2.3.3 The Common Active Users Feature: F_u

A common active user in BSs is defined as the user whose data traffic appears in all these BSs during a certain time period, e.g., 10 minutes. Since a user may have many traffic records during the time period, we only count the unique user ID on each BS as its user set. A large number of common active users in BSs indicates that active users frequently handover among these BSs, implying that these BSs are tightly coupled. A common active users in grids is defined as the user whose data traffic appears in all these grids. We analyze the mobile traffic data and derive the number of common active users of two BSs and two grids.

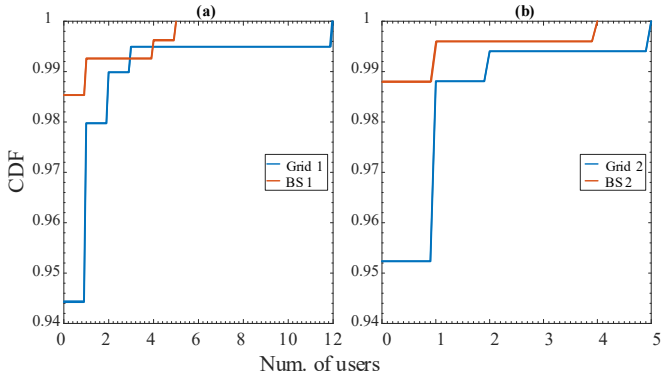


Figure 6.7: The common active user analysis.

The red line in Figure 6.7(a) shows the CDF of the number of common active users in BS 1 and its neighboring BSs in 10 mins. The blue line shows the CDF of the number of common active users in Grid 1 and its neighboring grids. The numbers of common active users are different under different network partitioning granularities. BS 1 shares common active users with about 1.34% of its neighboring BSs, and Grid 1 shares common active users with about 5.58% of its neighboring grids. As compared with the CDFs shown in Figure 6.7(b), we can see that both BSs and Grids at different locations have different CDFs of common active common users. These observations verify that the number of common active users can be used as a network feature for

network partitioning.

Based on the three network features, the distance function that quantifies the relationship between the ith and jth BSs is expressed as $D_{i,j} = \sum_{k=1}^{F} \alpha_k d_{i,j}^k$. Here, F is the number of features used in the distance calculation; α_k is the weight of the kth feature; $d_{i,j}^k$ is the distance calculated based on the kth feature.

6.3 Offline Machine Learning

The offline learning module consists of two parts: the network performance aware clustering analysis and the optimal BS cluster number prediction module.

6.3.1 Network Performance Aware Clustering

The objective of the network performance aware clustering analysis is to derive the optimal network partitioning solution under different mobile traffic conditions such that the analysis results can be utilized for training the prediction model. The network clustering analysis contains two parts: the network clustering and clustering solution evaluation.

6.3.1.1 Hierarchical Clustering Algorithm

Network partitioning aims to divide the entire network into sub-RANs. BSs in the same sub-RAN have tight connections with each other as compared to BSs in other sub-RANs. To partition the network, we use the HCA algorithm to agglomerate a set of objects into a hierarchy of clusters based on the distances of objects [44]. The advantage of HCA over other clustering algorithms such as K-medoids clustering [139] is that it does not require *a priori* knowledge of the number of clusters. Besides, HCA can flexibly adjust the number of clusters for different network scenarios. Since in network partitioning, the optimal number of BS clusters cannot be predefined due to mobile traffic dynamics. For instance, when the network is busy, networking functions need to be quickly processed. In this case, the network will be partitioned into more sub-RANs to achieve optimal networking performance. HCA allows us to select a

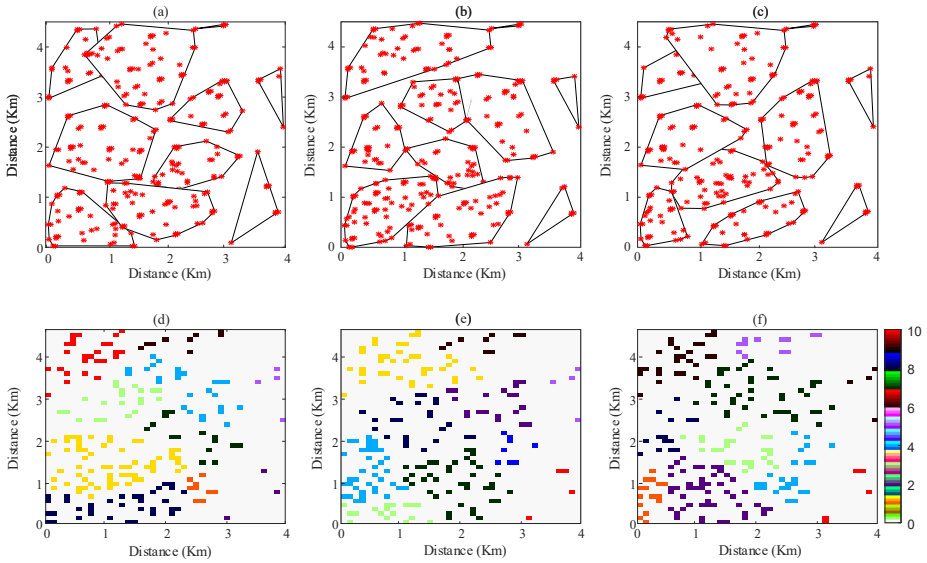


Figure 6.8: The visualization of network partitioning. (9 (a), (b) and (c): BS-based network partition at 8:00, 14:00 and 20:00, respectively; 9 (d), (e) and (f): grid-based network partition at 8:00, 14:00 and 20:00, respectively.)

high layer in the clustering tree without clustering the whole network again. Thus, the HCA algorithm is adopted in engineering the training data set for the prediction model.

Based on the features extracted from the mobile traffic data in Section 6.2.3, the distance between the ith and jth BSs is expressed as $D_{i,j} = \sum_{k=1}^{N} \alpha_k d_{i,j}^k$. Here, N is the number of features used in the distance calculation; α_k is the weight of the kth feature; $d_{i,j}^k$ is the distance calculated based on the kth feature. After a round of merging BSs, a few BSs will be grouped in the same cluster. $\mathcal{D}(C_m, C_n) = \min_{i \in C_m, j \in C_n} D_{i,j}$ is defined as the distance between the BS clusters C_m and C_n. Therefore, the distance between two BS clusters reflects the minimum inter-BS distance for BSs in different clusters.

The HCA algorithm is illustrated in Algorithm. 4. At the beginning, each BS is treated as an individual cluster. Based on the distance function, the tightness of the coupling between clusters can be evaluated. A shorter distance indicates a tighter coupling between the clusters. Based on the distances between clusters, the HCA algorithm iteratively merges clusters with the shortest distance into a new cluster.

After each round of merging clusters, the distances between clusters are recalculated. The merging process continues until all BSs are in one cluster. Once the clustering tree $\mathcal{C}_T$ is derived, we may select one layer in the tree to partition the network. For example, if the Kth layer of $\mathcal{C}_T$ is selected, the entire network is partitioned into K sub-RANs. For grid-based network partitioning, we use the grid-based feature set to build the distance function and cluster the grids. Figure 6.8 shows the visualization of network partitioning based on the BS-based and grid-based network partitioning solutions. We choose three time slots to partition the network, and partition the network into 10 sub-RANs, as shown in Figure 6.8(a), 6.8(b) and 6.8(c) corresponding to 8:00, 14:00, and 20:00, respectively.

Algorithm 4: The Hierachical Clustering Algorithm

Input : A feature set$\{F_d, F_t, F_u\}$, a cluster set $\mathcal{C}$;
Output: A cluster tree $\mathcal{C}_T$;
Initialize the layer L(0) = 0;
Initialize the sequence number m = 0;
while $|\mathcal{C}| > 1$ **do**
    Append($\mathcal{C}$) into $\mathcal{C}_T$;
    Find pair (c_r, c_s) with $\mathcal{D}(c_r, c_s) = \min \mathcal{D}(c_i, c_j)\ \forall c_i, c_j \subset \mathcal{C}, i \neq j$;
    m = m +1, L(m)=$\mathcal{D}(c_r, c_s)$;
    $c_{new} = c_r \cup c_s$;
    $\mathcal{C} = \mathcal{C} - \{c_r, c_s\}$;
    $\mathcal{C} = \mathcal{C} \cup c_{new}$;
Reverse the tree $\mathcal{C}_T$;
return $\mathcal{C}_T$;

6.3.1.2 Clustering Evaluation

After partitioning the network, we run the network function on the entire network and individual sub-RANs, respectively. Since network function is concurrently running in individual sub-RANs, we use the maximum computing time required in the sub-RANs to represent the network function computing time after network partitioning. Denote τ and $\{\hat{\tau_1}, \hat{\tau_2}, \cdots, \hat{\tau_K}\}$ as the computation time of the network function before and after the network partitioning, respectively. $\hat{\tau_i}$ is the computing time

of the ith sub-RAN. Hence, the computation performance incurred by the network partitioning solution is $\mathcal{V} = \frac{\max(\hat{\tau})}{\tau}$.

Denote ζ and $\{\hat{\zeta_1}, \hat{\zeta_2}, \cdots, \hat{\zeta_K}\}$ as the network function performance before and after network partitioning, respectively. $\hat{\zeta_i}$ is the networking performance of the ith sub-RAN. Then, the networking performance incurred by the network partitioning solution is $\mathcal{P} = \frac{\sum_{i=1}^{K}(\hat{\zeta_i})}{\zeta}$. In the network clustering, the networking performance is traded for the computation performance. It is desirable to limit the networking performance degradation when partitioning the network. Given $\mathcal{P}$ and $\mathcal{V}$, we derive the performance of network partitioning by calculating $\psi = (1-\omega)\mathcal{P} + \omega(1-\mathcal{V})$.

In the network clustering, multiple parameters may impact the performance of the clustering. These parameters include the number of clusters and the weights of different networking features. With a given ω, in order to guide the HCA algorithm to find the optimal network partitioning solution, we traverse all kinds of networking feature weights combinations. For each wights combination, we generate the corresponding HCA clustering tree and traverse the entire clustering tree to obtain the maximum ψ and the corresponding number of clusters. After comparing the maximum ψ from different clustering trees, we get the optimal number of BS clusters K, the so-called ground truth value. These results are transferred to the predictor for training the prediction model. The computational complexity to obtain ground truth values is extremely high, and thus driving the optimal network partitioning solution in real-time is not practical. To solve this problem, we perform the network partitioning in two steps. First, we develop BS cluster number prediction methods that predict the optimal number of the BS cluster under different traffic conditions. Second, we design a real time network partitioning algorithm based on the prediction result.

6.3.2 Cluster Number Prediction Model

The cluster number predictor aims to predict the optimal number of BS clusters in realtime based on the traffic condition. In order to accurately predict the optimal

number of BS clusters, we choose three widely used machine learning algorithms: Least-squares regression [143], SVM regression [144] and ANN [145], to build three predictors. The least-squares regression is one the simplest prediction algorithms, and we use its performance as the baseline. Both the SVM regression and ANN are widely used prediction models and have been applied in wireless networks [146, 147]. The main limitation of the ANN model is the requirement of a big training data set. Otherwise, we may get a poor prediction result. For SVM regression, the limitation is that we need to determine some key parameters. Since we do not have any priori knowledge about the relationship between the traffic loads and the optimal BS cluster number, it is hard to obtain the best value of these parameters.

6.3.2.1 Input Generation

We define the duration of a time slot to be 10 minutes. Denote γ_i as the accumulative traffic load in the ith BS in one time slot. Assume there are n BSs in the network. Then, we use $\Gamma = \{\gamma_1, \gamma_2, \cdots, \gamma_n\}$ to represent the network traffic load in one time slot. The normalized Γ is one of the inputs to the ANN. Then, we use the network performance aware clustering analysis method to derive the optimal BS cluster numbers at different time slots. The optimal BS cluster number K acts as the ground truth value.

6.3.2.2 ANN predictor

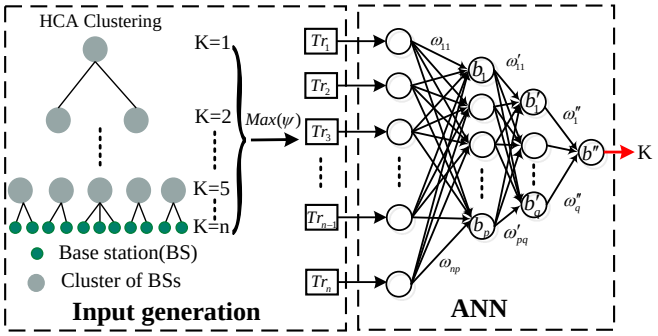


Figure 6.9: The structure of the neural network predictor.

We adopt the back-propagation neural network as the prediction algorithm. As illustrated in Figure 6.9, our neural network consists of the input layer, two hidden layers, and the output layer. We assign a sigmoid function $\varphi(x)$ for the neurons in the hidden layers and a purelin function $I(x)$ for the output layer. The output of the prediction is expressed as

$$K = I(\sum_{q} \omega_q^{''} \varphi(\sum_{p} \omega_{pq}^{'} \varphi(\sum_{n} \gamma_n \omega_{np} + b_p) + b_q^{'}) + b^{''}), \tag{6.3}$$

where $b^{''}$, $b_q^{'}$ and b_p are the biases associated with the neurons in different layers. $\omega_q^{''}$, $\omega_{pq}^{'}$ and ω_{np} denote the weights associated with the inputs of different layers.

6.3.2.3 Least-squares regression predictor

As defined in the previous section, the training data set is composed of (Γ,K), where Γ is the traffic load for one time slot, and K is the optimal clustering number. A least-squares regression model can be represented by:

$$K = f(\Gamma, \beta) + \epsilon \tag{6.4}$$

$$\sum \epsilon^2 = \sum_{i=1}^{n} (K_i - f(\Gamma_i, \beta))^2 \tag{6.5}$$

Our goal is to minimize $\sum \epsilon^2$. Here, ϵ is the error between the prediction value and the grand truth value. β is the coefficient set we want to learn.

6.3.2.4 SVR predictor

The support vector machine regression (SVR) can be represented by

$$f(\Gamma) = \sum_{i=1}^{n} (\alpha_n - \alpha_n^*) G(\Gamma_i, \Gamma) + b. \tag{6.6}$$

Here, $G(\Gamma_i, \Gamma_j)$ is the kernel function. Linear, Gaussian, Polynomial kernels are widely used. In this paper, we adapt the Linear and Gaussian kernel to build the

SVR predictor. $f(\Gamma)$ is the predicted optimal BS cluster number. α_n and α_n^* are two non-negative multipliers.

The performance of networking partitioning depends on the distance function, which quantifies the relationships between BSs. Since mobile traffic is highly dynamic, the distance functions should be revised according to mobile traffic conditions and the prediction model should be retrained periodically. As shown in Figure 6.10, mobile traffic loads periodically change in both the macro BS(MBS) and small cell BS (SCBS). Leveraging the diurnal pattern of the mobile traffic, we periodically validate our prediction model to ensure the accurate prediction. Although the number of traffic patterns observed from the dataset is limited, the relationships between BSs are characterized by three network features in the paper rather than solely by the traffic pattern. For example, the number of common users in BSs is adopted as one of the network features. Owing to the user mobility, the common users in BSs change tremendously over time. The network partition solution should consider such changes in the network. Therefore, we need an online algorithm to dynamically generate network partitions for network management. To validate the predictor, the optimal number of BS clusters K^g obtained from the clustering analysis will be compared with the predicted BS cluster numbers K. The prediction model will be retrained if the performance of the network partitioning with K clusters is less than 90% of that with K^g clusters. During the retraining process, the latest mobile traffic record data will be integrated into the dataset for re-training the prediction model.

6.4 Online Virtual Network Function Placement

The online network function placement module consists of the online network partitioning and the network function placement. We partition the network into K sub-RANs, and we place the network functions in each sub-RAN.

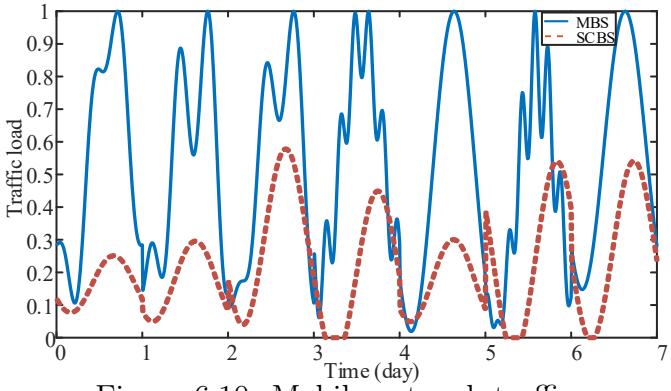


Figure 6.10: Mobile network traffic.

6.4.1 Online Network Partitioning

The computational complexity of clustering algorithms is vital to achieve the realtime network function placement. As compared with the HCA algorithm, the K-medoids algorithm has lower computational complexity. The complexity of HCA is $o(n^2 * logn)$ [148], where n is the total number of BSs in the network. The complexity of the K-medoids is $o(ikn)$. Here, k is the number of clusters and i is the number of iterations required for convergence. In addition, since the optimal number of BS clusters has been predicted by the predictors, there is no need to generate the clustering tree. Hence, we adopt the K-medoids algorithm as our online network partitioning algorithm. The pseudo code of the K-medoids algorithm is presented in Algorithm 5.

Algorithm 5: The K-medoids Clustering Algorithm

Input : A feature set $\{F_d, F_t, F_u\}$, number of clusters k, a cluster set $\mathcal{C}$;
Output: A cluster $\mathcal{C}_T$;

```
Randomly select k points in the cluster set as k medoids;
while k medoids changes do
    Associate each point to the closest medoid and formulate k clusters C_k;
    for i=1:k do
        Find the point x_i in C_i that minimizes sum_{j in C_i}^{|C_i|} ||x_i - x_j||_2^2;
        Set the point as the new medoid;
return C_T;
```

Once the BSs are divided into clusters, each BS cluster is optimized independently with the network functions. The network partitioning solution will be different when

the clustering performance is evaluated under different network functions. In this paper, we implement a traffic load balancer (TLB) as the network function to evaluate the performance of the clustering [36].

6.4.2 Network Function: Traffic Load Balancer

TLB solves the following optimization problem [36]:

$$\begin{aligned} \max_{x,y} & \sum_{i\in\mathcal{U}} U(R_i) = \sum_{i\in\mathcal{U}} U(\sum_{j\in(B)} y_{ij}c_{ij}) \\ s.t. \quad & \sum_{j\in\mathcal{B}} x_{xj} = 1, \forall i \in \mathcal{U} \\ & \sum_{i\in\mathcal{U}} y_{ij} \leq 1, \forall j \in \mathcal{B} \\ & y_{ij} \leq x_{ij}, x_{ij} \in \{0,1\}, \forall i \in \mathcal{U}, \forall j \in \mathcal{B}. \end{aligned} \tag{6.7}$$

The objective is to maximize the aggregated utilities of all users. The problem involves finding the indicators x_{ij} corresponding to the user-BS association. $x_{ij} = 1$ indicates that the ith user is associated with the jth BS. y_{ij} is the corresponding resource allocation that maximizes the aggregate utility function. $c_{ij} = log_2(1 + SINR_{ij})$ is the achievable data rate between the ith user and the jth BS.

The procedure of the TLB function is illustrated in Figure 6.11. Users measure their achievable data rates from different BSs and report the measurements to their serving BSs. The BSs collect these measurements and forward them to the network controller. The traffic load optimization algorithm runs in the network controller and optimizes the user-BS association parameter denoted as μ_j^*. The network controller broadcasts the user-BS association parameters via BSs to users. Upon receiving the user-BS association parameters, users select BSs to maximize their own utility.

We run the TLB function on the entire network and individual sub-RANs, respectively. Then, we calculate the $U(R_i)$ and $U(\hat{R}_i)$, which are the ith user's utility before and after network partitioning, respectively. The networking performance of network

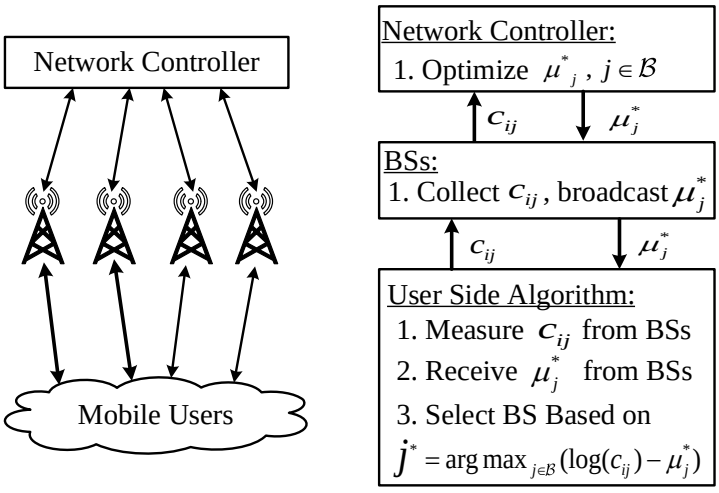


Figure 6.11: The traffic load balancing function.

partitioning can be derived by calculating $\mathcal{P} = \frac{\sum_{i\in\mathcal{U}} U(\hat{R}_i)}{\sum_{i\in\mathcal{U}} U(R_i)}$. Since the TLB function is concurrently running in individual sub-RANs in the network controller, we use the maximum computation time required in the sub-RANs to represent the computing time after network partitioning. Hence, the computation performance of network partitioning is $\mathcal{V} = \frac{\max(\hat{\tau})}{\tau}$. Given $\mathcal{P}$ and $\mathcal{V}$, we derive the performance of network partitioning by calculating $\psi = (1-\omega)\mathcal{P} + \omega(1-\mathcal{V})$. Here, $0 < \omega < 1$ is a parameter, which can be selected by the network operator. Given ω, the clustering solution that maximizes ψ is the optimal network partitioning solution.

6.5 Performance Evaluation

In this section, we evaluate the performance of the data-driven network optimization framework through data trace-driven network simulations.

6.5.1 Mobile Traffic Dataset and Simulation Settings

Our wireless big data are obtained from an anonymized data trace collected from an operating mobile network with a duration of two weeks. Each entry in the trace includes the BS ID, geo-location of the BS, the user ID, and start-end time of the data connection. The data trace contains 2.8 billion tuples of the information collected from over 10000 BSs and 500000 users. The greatest density of BSs in the dataset

is 15 BSs per 100 square meters, which can be considered as an ultra-dense mobile network [35]. This large amount of data ensures that our analysis and modeling are credible.

For network partitioning and predicting the optimal number of BS clusters, we define the duration of a time slot to be 10 minutes. Then, there are 144 time slots per day. Γ_i is defined as the traffic load set of the network at the ith time slot. In the prediction model training process, 80% of mobile traffic data are used to train the prediction model, and the remaining 20% are used to evaluate the predictor.

On simulating the traffic load balancing function, we set $\omega = 0.5$. Transmit powers of macro and small cell BSs are $P1 = 46$dBm and $P2 = 35$dBm, respectively. On modeling the propagation environment, we adopt a path loss $L(d) = 34 + 40log(d)$ and $L(d) = 38 + 30log(d)$ for macro and small cell BSs, respectively. We adopt the log normal shadowing fading with a standard deviation $\sigma_s = 8$dB. The thermal noise power is assumed to be $\sigma^2 = -104$dBm.

6.5.2 Networking Feature Evaluation

In this section, we justify the effectiveness of the networking features. As shown in Table 6.3, we partition the network using the HCA algorithm with different combinations of the three features. Considering the traffic load variations over time, we analyze the performance of the network partitioning in three different time slots. In this simulation, the number of sub-RANs is 10 and ω is set to 0.5.

We compare each individual feature first. According to Table 6.3(numbers with color are the optimal values in the column), the network partitioning solution based on F_d allows the TLB function to achieve the maximum ψ; this proves that geo-distance is a requisite feature. Moreover, the network partitioning solution generated by using F_t can always allow the TLB function achieve the maximum $\mathcal{P}$. However, the computation performance is sacrificed. The reason is that when we partition the network only based on F_t, a big sub-RAN containing most of the BSs in the network

Table 6.3: Networking feature evaluation results

	8:00			**14:00**			**20:00**		
Features	$\mathcal{P}$	$\mathcal{V}$	ψ	$\mathcal{P}$	$\mathcal{V}$	ψ	$\mathcal{P}$	$\mathcal{V}$	ψ
F_d	0.9472	0.1855	0.8809	0.9265	0.2213	0.8526	0.9326	0.1341	0.8992
F_t	0.9892	0.9819	0.5036	0.9398	0.8936	0.5231	0.9697	0.9640	0.5029
F_u	0.8678	0.2572	0.8053	0.8302	0.2161	0.8071	0.8695	0.2896	0.7900
$F_d + F_t$	0.9461	0.1711	0.8875	0.9237	0.2243	0.8497	0.9413	0.2234	0.8589
$F_d + F_u$	0.9473	0.2156	0.8659	0.9237	0.1981	0.8628	0.9370	0.1942	0.8714
$F_t + F_u$	0.8737	0.2749	0.7994	0.8467	0.2287	0.8090	0.8697	0.2584	0.8056
$F_d+F_t+F_u$	0.9459	0.1482	0.8988	0.9364	0.1848	0.8758	0.9400	0.1390	0.9005

will be generated in the partitioning solution. Since the complexity of running TLB in this big sub-RAN is still very high, partitioning the network solely based on F_t is not appropriate. The partitioning solution solely based on F_u will lead to a poorer network performance. However, when combined with other features, the obtained network partitioning solution can obviously improve $\mathcal{V}$ of the TLB function. For instance, when the network is partitioned based on $\{F_d, F_t\}$, the computation performance of the TLB function is 0.1711. When the network is partitioned according to $\{F_d, F_t, F_u\}$, the computation performance of the TLB function is 0.1482. Therefore, by introducing a new network feature F_u, the computation performance of the TLB function improves 0.0229.

When the network is partitioned with a combination of three networking features, the network partitioning solution can always enable the TLB function to achieve the maximum ψ. In addition, the partitioning solution can also significantly improve the computation performance $\mathcal{V}$ of the network function in all of the time slots. This indicates that when network is partitioned with the combination of three networking features, the network partitioning solution does not allow a sub-RAN with a large number of BSs.

6.5.3 Network Function Performance

We evaluate the network clustering performance in three steps. First, we study the impact of the number of sub-RANs K on the networking and computation per-

formance. Second, we analyze the networking and computation performance on each sub-RAN with two network partitioning granularities. Third, we investigate the networking and computation performance of the network function with the partitioning solution obtained by using the prediction model.

Fig. 6.12 shows the impact of the number of sub-RANs on the performance of the TLB function. As the number of sub-RANs, K, increases, the computation performance is significantly enhanced. The larger K, the less computing time required for executing the TLB function. However, when $K > 8$, the improvement of the computation performance diminishes. The networking performance drops as the number of sub-RANs increases. A larger number of sub-RANs may constrain more users from being served by certain BSs. As a result, the aggregated network utility is reduced. When K increases from 1 to 10, the networking performance $\mathcal{P}$ drops 10%. When K increases from 10 to 20, it only drops 4%. This indicates that the performance drop also slows down when K is larger than a certain number, e.g., 10 in the simulation. Based on network management strategies, network operators can determine the tradeoff between the computation performance and the networking performance.

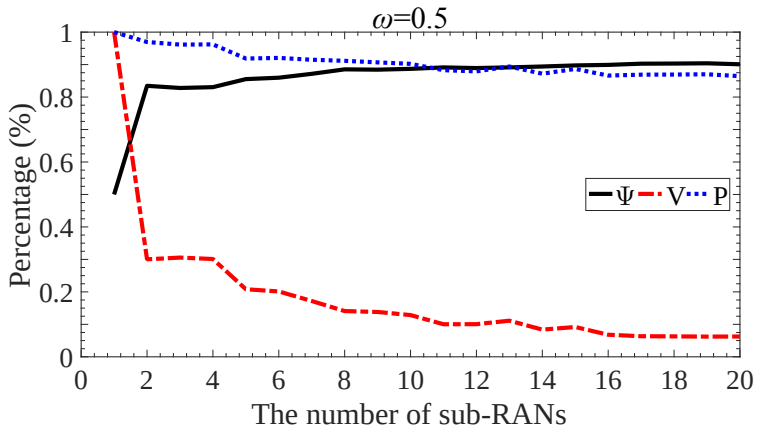


Figure 6.12: Network partitioning performance versus the number of sub-RANs.

We next evaluate the networking and computation performance on each sub-RAN, respectively. Fig. 6.13 shows cumulative distribution functions (CDFs) of the compu-

tation and networking performance in individual sub-RANs. Fig. 6.13(a) shows CDFs of the computation performance with BS-based and grid-based network partitioning, respectively. Let $\mathcal{B}_s$ be the set of sub-RANs. Denote $\nu_i = \frac{\tau_i}{\tau}$ as the computation performance of the ith sub-RAN. Based on ν_i, $\forall i \in \mathcal{B}_s$, we derive the CDF of the computation performance in individual RANs. As shown in Fig. 6.13(a), with BS-based network partitioning, 70% of the sub-RANs save more than 90% computing time while the remaining 30% sub-RANs save 80% to 90% computing time after network partitioning. This proves that network partitioning significantly improves computation performance of the network function. With grid-based network partitioning, 90% of the sub-RANs save more than 90% computing time while the remaining 10% sub-RANs save 85% to 90% computing time after network partitioning. This proves that grid-based network partitioning can also improve the computation performance significantly.

Fig. 6.13(b) shows the CDFs of the networking performance in individual sub-RANs. For the TLB function, $\mathcal{U}_i$ and $\hat{\mathcal{U}}_i$ are denoted as the set of the users associated with BSs in the ith sub-RAN before and after network partitioning, respectively. Let p_i be the networking performance of the ith sub-RAN. Then, $p_i = \frac{\sum_{j \in \hat{\mathcal{U}}_i} U(\hat{R}_j)}{\sum_{j \in \mathcal{U}_i} U(R_j)}$. Based on p_i, $\forall i \in \mathcal{B}_s$, we derive the CDF of the networking performance of individual sub-RANs. As shown in Fig. 6.13(b), with BS-based network partitioning, the networking performance only decreases less than 10% in more than 40% of sub-RANs. Besides, about 20% of sub-RANs increase their networking performance. Only 10% of sub-RANs suffer from network partitioning in terms of the networking performance. The networking performance of these sub-RANs decreases up to 40%. This analysis shows that network partitioning can provision computation efficient network management at a low cost of the networking performance. With grid-based network partitioning, the networking performance only decreases less than 10% in more than 35% of sub-RANs. Besides, about 10% of sub-RANs increase their networking performance. About 20%

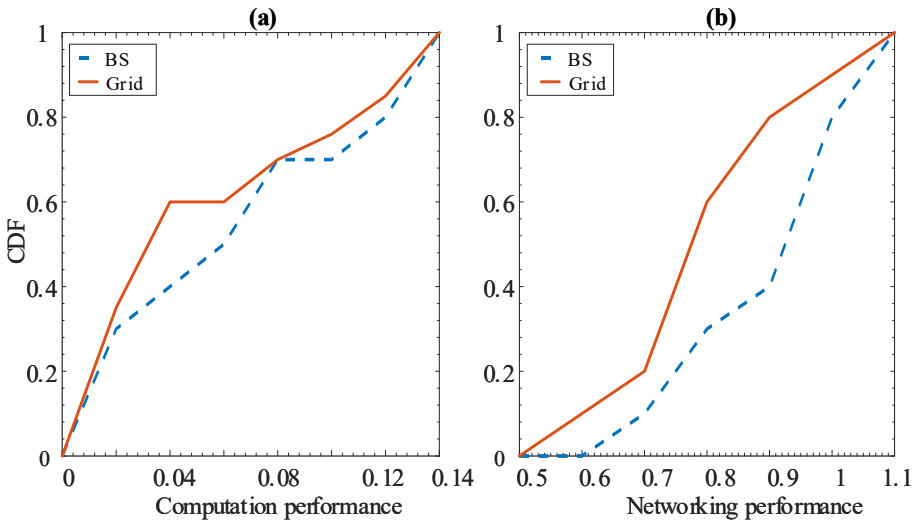


Figure 6.13: The computation and networking performance analysis.

of sub-RANs suffer from network partitioning in terms of the network performance, and they lose more than 30% networking performance.

After training the prediction model, we predict the optimal BS cluster number K for each time slot in the test data set. We then partition the network into K sub-RANs and place the TLB function in each sub-RAN. Fig. 6.14 shows the CDFs of computation and networking performance of the TLB function for all time slots. Fig. 6.14(a) presents the computation performance. With BS-based network partitioning, it shows that using the predicted K to partition the network, the computation time of the network function is less than 10% of that of the optimal networking mechanism in about 95% of the simulated time slots. Here, the optimal networking mechanism is defined as the network function executed on the entire radio access network without any network partitioning. This result proves that the proposed framework can effectively improve computation performance of the network function.

Fig. 6.14(b) presents the networking performance, and shows that BS-based network partitioning can achieve at least 85% of the optimal network performance in all simulated time slots. In 80% of the simulated time slots, our framework can achieve 90% of the optimal network performance. Therefore, the proposed framework can significantly reduce the computation time of the networking mechanism at a low cost

of the networking performance. With grid-based network partitioning, we can achieve at least 80% of the optimal network performance in all simulated time slots. In 70% of the simulated time slots, our framework can achieve 90% of the optimal network performance.

With grid-based network partitioning, the BSs within the same grid cannot be grouped into different sub-RANs. As a result, grid-based network partitioning has less flexibility in terms of grouping BSs than BS-based network partitioning.

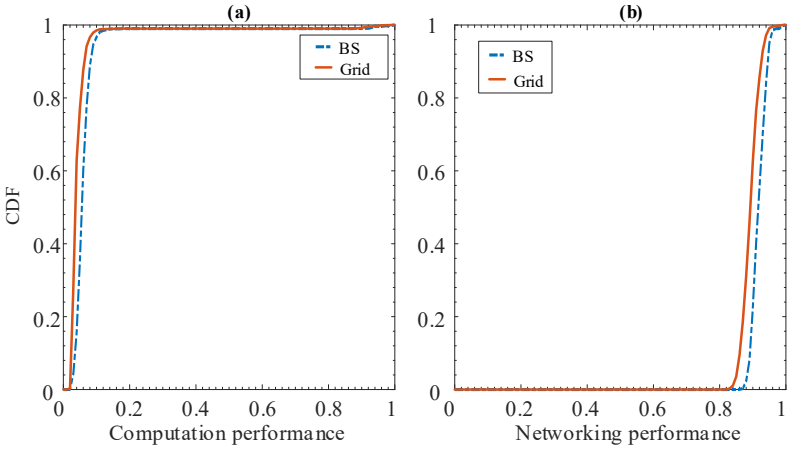


Figure 6.14: The CDF of system performance.

6.5.4 Prediction Model Performance

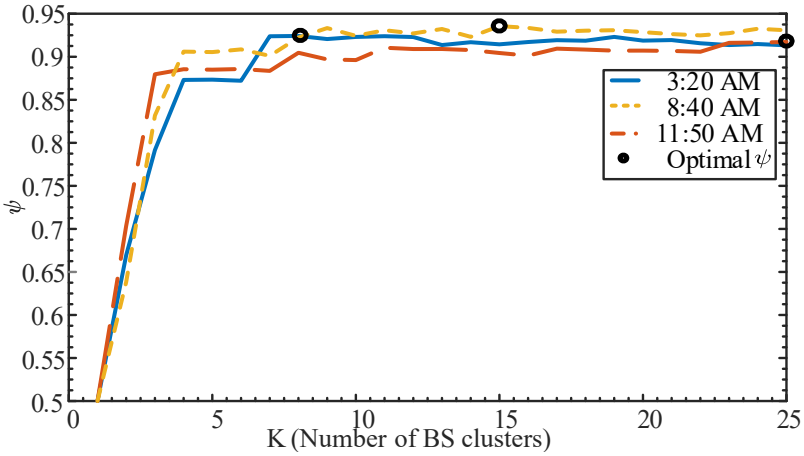


Figure 6.15: The performance of the prediction model.

To evaluate the performance of the cluster number predictors, we first analyze the impact of the cluster number on the performance of the network functions. Fig. 6.15 shows the performance of our framework under three different traffic cogitations. 3:20 AM, 8:40 AM, and 11:50 AM represent low, medium, and high traffic load layer, respectively. We can observe that the optimal cluster number, K^*, with the maximum ψ for different time slots are 8, 15, and 25, respectively. Denote $\hat{K}$ as the number of BS clusters predicted from the ANN predictors, we can see that ψ with the prediction errors $|\hat{K} - K^*| = 1$, $|\hat{K} - K^*| = 2$, and $|\hat{K} - K^*| = 3$ will only be reduced on average by 0.5%, 1.5%, and 2%, respectively. In this case, we consider $\hat{K}$, which satisfies $|\hat{K} - K^*| = 1$, acceptable.

To evaluate the accuracy of our prediction models, we define the predicted result $\hat{K}$, which satisfies $|\hat{K} - K^*| = 0$ as the accurate prediction. The $\hat{K}$, which satisfies $|\hat{K} - K^*| <= 1$, as the acceptable prediction. The accuracy rate is defined as the number of correct predictions divided by the number of all predictions. Fig. 6.16(a)

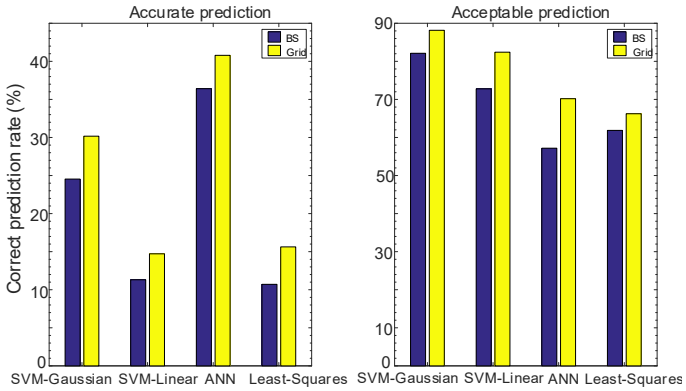


Figure 6.16: The prediction error of different predictors.

shows the accurate prediction results performed by three cluster number predictors in BS-based and grid-based network partitioning. The SVM regression model always obtains a higher accuracy rate than the Least-squares regression model. The ANN predictor achieves the highest accuracy rate, 36%. The accurate prediction rates of all the BS cluster number predictors in grid-based network partitioning is higher than

that in BS-based network partitioning. Acceptable prediction results are shown in Fig. 6.16(b). The SVM regression model always obtains the best accuracy rate in both BS-based and grid-based network partitioning. In BS-based network partitioning, the ANN predictor has the worst prediction accuracy rate, 57%. However its accuracy rate increases to 70% in grid-based network partitioning. The reason is that the dimensions of the input vector for the training model are reduced in grid-based network partitioning, and thus the prediction performance can be improved.

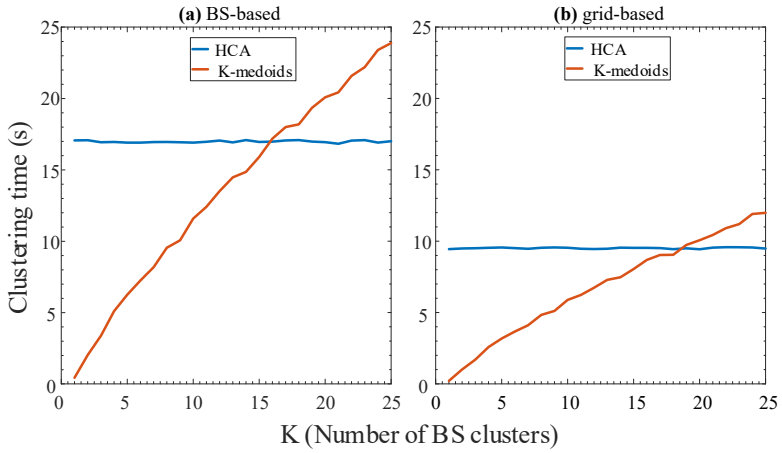


Figure 6.17: The computation time of clustering algorithms.

6.5.5 Clustering Algorithm Performance

Fig. 6.17 shows the comparison of the computation time of HCA and K-medoids algorithms in both BS-based and grid-based network partitioning. The computation time for clustering BSs using the HCA algorithm does not change with the number of BS clusters. This is because the HCA algorithm needs to generate the entire clustering tree no matter what the value of K is. The computation time of the K-medoids algorithm is linearly proportional to the number of BS clusters. When the number of BS clusters is small, the computation time of the K-medoids algorithm is much less than that of the HCA algorithm.

As compared to BS-based network partitioning, grid-based network partitioning saves about 43% and 46% computation time with the HCA and K-mean algorithms,

respectively.

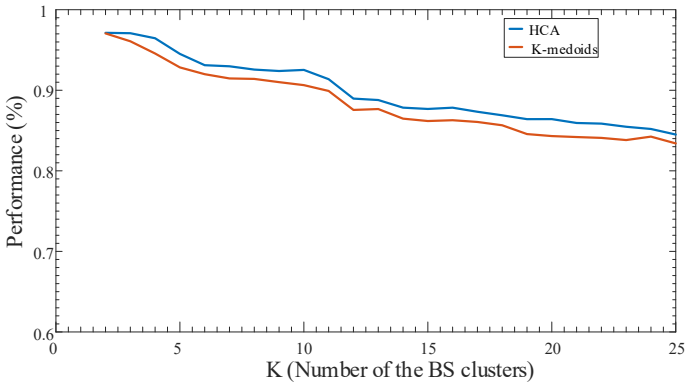


Figure 6.18: The performance of clustering algorithms.

Fig. 6.18 shows the performance comparison of different clustering algorithms. The performance of the HCA algorithm is only slightly better than that of the K-medoids algorithm. The HCA algorithm has a better performance because it merges the clusters with the minimal inter-clusters distance in building the clustering tree. However, the computational complexity of the K-medoids algorithm is significantly lower than that of the HCA algorithm. Therefore, the K-medoids algorithm is more suitable for realtime network partitioning.

CHAPTER 7: CONCLUSION

7.1 Completed Work

In this dissertation, two deep learning driven optimization frameworks, Pearl (a deep learning driven video compression framework) and DAVE (dynamic adaptive video encoding framework), are proposed to improve the video visual quality and reduce the network bandwidth usage of video transmissions for resource-hungry video streaming applications. Three novel schemes, an improved frame-level offloading decision scheme, a new online model updating scheme, and a smart task allocation decision scheme, are proposed to satisfy the high computation complexity and low latency requirement of MAR applications. Furthermore, a data-driven network optimization framework is proposed to enable realtime and efficient ultra-dense network optimization.

For video delivery applications, we have proposed a deep learning driven compression framework named Pearl, that utilizes the power of deep learning to compress UHD videos. An optimal compact representation from the origin UHD videos is learned with channel-based super resolution models. The super resolution model is used to reconstruct a UHD video from a low-resolution video. Pearl can compress up to 95% of video data size during the video transmission, which significantly saves network bandwidth resources and reduces the network transmission latency.

The video perceptual quality of current real-time video streaming systems is normally sacrificed to meet the very low-latency requirement, and traditional adaptation strategies in live video streaming systems cannot be directly applied to the real-time video streaming system. To overcome these challenges, a new real-time video streaming protocol, DAVE, is proposed. DAVE uses a reinforcement learning model to learn

the optimal video encoding configurations under different network conditions, and a super resolution algorithm is applied in DAVE to improve the video perceptual quality. Extensive evaluations based on both simulation and a real testbed show that DAVE substantially improves the performance of existing real-time video streaming systems in terms of improved video perceptual quality and reduced end-to-end latency.

For mobile augmented reality applications, we have proposed a smart-decision framework which combines the advantages of the on-device mobile AR system and the edge-based mobile AR system to achieve real-time recognition tasks. High computation complexity tasks will be offloaded to the edge servers. Low complexity tasks will be executed on the mobile devices or the edge server depending on the network latency. We design a cache and matching system to enhance the performance of mobile AR applications when the on-device deep learning models have poor performance.

Executing object detection on mobile devices faces one challenge: poor user QoE caused by detection-failure. Experimental results from our implemented testbed indicate that existing MAR systems that apply offloading and tracking algorithms cannot be used to improve the poor user QoE caused by detection-failure. We addressed this challenge by exploring the characteristics of detection-failure in MAR applications. A new frame-level offloading decision algorithm and the first online model replacement scheme for MAR systems are proposed. To the best of our knowledge, this is the first work that addresses this problem. Our proposed designs are complementary to existing offloading decision optimization schemes. It can substantially reduce the detection-failure rate and improve user QoE.

For ultra-dense mobile networks, we have proposed a wireless big data-driven network function placement framework, which integrates wireless big data analysis and network functions, to enable real-time and efficient network management. The proposed framework consists of a data processing module, an offline machine learning module, and an online network function placement module. We employ big data

analytics to extract useful information and use machine learning to achieve near real-time VNF placement for ultra-dense wireless networks. Our proposed framework significantly reduces the computational complexity of network functions.

7.2 Future Work

Based on our analysis of resource-hungry applications, two issues can be considered in future work:

- More and more AI-enabled applications are deployed on mobile devices. However, to apply AI-enabled features, high computation complexity tasks (DNN models) need to be frequently executed. These model executions drain the battery on mobile devices very fast, which significantly reduces the experience time of mobile applications. These applications can also be defined as resource-hungry applications. Offloading the high computation complexity tasks to the edge server can save energy on mobile devices. However, additional transmission latency degrades user QoE. We can train an AI agent that will decide where to execute the task to balance the energy consumption and latency of these resource-hungry applications.

- DNN models usually have a very high computation complexity, which is very challenging for mobile devices. Model compression can significantly reduce the model computation complexity which brings a faster execution speed and less energy consumption. We can adopt model compression to further enhance the performance of resource-hungry applications.

7.3 Published and Submitted Work

The following list is a summary of my publications:

- Siqi Huang, Haoxin Wang, and Jiang Xie, “Improving the effectiveness of object detection for edge-based mobile augmented reality," (submitted).

- Siqi Huang, Jiang Xie, and Muhana Magboul Ali Muslam, “Configuration adaptive video encoding for real-time video streaming applications," (submitted).

- Siqi Huang, Jiang Xie, and Muhana Magboul Ali Muslam, “A cloud computing based deep compression framework for UHD video delivery," *IEEE Transactions on Cloud Computing*, 2021.

- Siqi Huang and Jiang Xie, “DAVE: Dynamic adaptive video encoding for real-time video streaming applications," *in Proceedings of IEEE International Conference on Sensing, Communication and Networking (SECON)*, July 2021.

- Siqi Huang and Jiang Xie, “Pearl: A fast deep learning driven compression framework for UHD video delivery," *in Proceedings of IEEE International Conference on Communications (ICC)*, June 2021.

- Siqi Huang, Tao Han, and Jiang Xie, “A smart-decision system for real-time mobile AR applications," *in Proceedings of IEEE Global Communications Conference (GLOBECOM)*, December 2019.

- Siqi Huang, Tao Han, and Nirwan Ansari, “Data-driven network Optimization in ultra-dense radio access networks," *in Proceedings of IEEE Global Communications Conference (GLOBECOM)*, December 2017.

- Siqi Huang, Tao Han, and Nirwan Ansari, “Big-data-driven network partitioning for ultra-dense radio access networks," *in Proceedings of IEEE International Conference on Communications (ICC)*, June 2016.

- Siqi Huang, Xueqing Huang, and Nirwan Ansari, “Budget-aware video crowdsourcing at the cloud-enhanced mobile edge," *IEEE Transactions on Network and Service Management*, 2021.

- Tapang Daniel Kanba, Siqi Huang, and Xueqing Huang, “QoE-based server selection for mobile video streaming," *in Proceedings of IEEE/ACM Symposium on Edge Computing (SEC)*, November, 2020.

- Qiang Liu, Siqi Huang, Johnson Opadere, and Tao Han, “An edge network orchestrator for mobile augmented reality," *in Proceedings of IEEE Conference on Computer Communications (INFOCOM)*, April 2018.

- Qiang Liu, Siqi Huang, and Tao Han, “Fast and accurate object analysis at the edge for mobile augmented reality," *in Proceedings of the Second ACM/IEEE Symposium on Edge Computing (SEC)*, December 2017.

- Qiang Liu, Siqi Huang, Yang Deng, and Tao Han, “Demo abstract: MExR: Mobile edge resource management for mixed reality applications," *in Proceedings of IEEE Conference on Computer Communications Workshops (INFOCOM)*, May 2017.

REFERENCES

[1] "Cisco visual networking index: Global mobile data traffic forecast update, 2015–2020," Feb. 2016. White Paper.

[2] P. Casas, A. D'Alconzo, P. Fiadino, A. Bär, A. Finamore, and T. Zseby, "When YouTube does not work, Analysis of QoE-relevant degradation in Google CDN traffic," *IEEE Transactions on Network and Service Management*, vol. 11, no. 4, pp. 441–457, 2014.

[3] J. Jiang, V. Sekar, and H. Zhang, "Improving fairness, efficiency, and stability in HTTP-based adaptive video streaming with festive," *IEEE/ACM Transactions on Networking (ToN)*, vol. 22, no. 1, pp. 326–340, 2014.

[4] E. Ramadan, A. Narayanan, Z. Zhang, R. Li, and G. Zhang, "Big cache abstraction for cache networks," in *Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS)*, pp. 742–752, 2017.

[5] H. Mao, R. Netravali, and M. Alizadeh, "Neural adaptive video streaming with pensieve," in *Proceedings of ACM Special Interest Group on Data Communication*, pp. 197–210, 2017.

[6] H. Yeo, Y. Jung, J. Kim, J. Shin, and D. Han, "Neural adaptive content-aware Internet video delivery," in *Proceedings of 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI))*, pp. 645–661, 2018.

[7] G. Toderici, D. Vincent, N. Johnston, S. Jin Hwang, D. Minnen, J. Shor, and M. Covell, "Full resolution image compression with recurrent neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5306–5314, 2017.

[8] C. V. Forecast, "Cisco visual networking index: Forecast and trends, 2017–2022," *White paper, Cisco Public Information*, 2019.

[9] M. Stonebraker, U. Çetintemel, and S. Zdonik, "The 8 requirements of real-time stream processing," *ACM Sigmod Record*, vol. 34, no. 4, pp. 42–47, 2005.

[10] B. Jansen, T. Goodwin, V. Gupta, F. Kuipers, and G. Zussman, "Performance evaluation of WebRTC-based video conferencing," *ACM SIGMETRICS Performance Evaluation Review*, pp. 56–68, 2018.

[11] L. De Cicco, S. Mascolo, and V. Palmisano, "Feedback control for adaptive live video streaming," in *Proceedings of ACM Conference on Multimedia Systems (MMsys)*, pp. 145–156, 2011.

[12] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson, "A buffer-based approach to rate adaptation: Evidence from a large video streaming service," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 187–198, 2015.

[13] T. Stockhammer *et al.*, "Dynamic Adaptive Streaming over HTTP—Design Principles and Standards," in *Proceedings of the Second Annual ACM Conference on Multimedia Systems*, pp. 2–4, 2011.

[14] "HTTP Live Streaming." `https://developer.apple.com/streaming/`.

[15] J. C. Lin and S. Paul, "RMTP: A reliable multicast transport protocol," in *Proceedings of IEEE Conference on Computer Communications*, vol. 3, pp. 1414–1424, 1996.

[16] "WebRTC: Real-time communication for the web." `https://webrtc.org/`.

[17] L. De Cicco, G. Carlucci, and S. Mascolo, "Experimental investigation of the google congestion control for real-time flows," in *Proceedings of the ACM SIGCOMM Workshop on Future Human-centric Multimedia Networking*, pp. 21–26, 2013.

[18] M. Nagy, V. Singh, J. Ott, and L. Eggert, "Congestion control using FEC for conversational multimedia communication," in *Proceedings of the 5th ACM Multimedia Systems Conference*, pp. 191–202, 2014.

[19] H. Alvestrand, S. Holmer, and H. Lundin, "A google congestion control algorithm for real-time communication on the world wide web," *IETF Internet Draft*, 2013.

[20] `https://www.a9.com/what-we-do/visual-search.html/`.

[21] `https://www.microsoft.com/en-us/hololens/apps/holotour`.

[22] S. Gammeter, A. Gassmann, L. Bossard, T. Quack, and L. V. Gool, "Server-side object recognition and client-side object tracking for mobile augmented reality," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR) - Workshops*, pp. 1–8, 2010.

[23] T. Y.-H. Chen, L. Ravindranath, S. Deng, P. Bahl, and H. Balakrishnan, "Glimpse: Continuous, real-time object recognition on mobile devices," in *Proceedings of ACM Conference on Embedded Networked Sensor Systems (SenSys)*, pp. 155–168, 2015.

[24] U. Drolia, K. Guo, and P. Narasimhan, "Precog: Prefetching for image recognition applications at the edge," in *Proceedings of IEEE/ACM Symposium on Edge Computing (SEC)*, pp. 1–13, 2017.

[25] W. Zhang, S. Lin, F. H. Bijarbooneh, H. F. Cheng, and P. Hui, "CloudAR: A cloud-based framework for mobile augmented reality," in *Proceedings of Thematic Workshops of ACM Multimedia*, pp. 194–200, 2017.

[26] D. Wagner, D. Schmalstieg, and H. Bischof, "Multiple target detection and tracking with guaranteed framerates on mobile phones," in *Mixed and augmented reality, 2009. ISMAR 2009. 8th IEEE international symposium on*, pp. 57–64, IEEE, 2009.

[27] D. Wagner, G. Reitmayr, A. Mulloni, T. Drummond, and D. Schmalstieg, "Real-time detection and tracking for augmented reality on mobile phones," *IEEE transactions on visualization and computer graphics*, vol. 16, no. 3, pp. 355–368, 2010.

[28] M. Shoaib, S. Venkataramani, X.-S. Hua, J. Liu, and J. Li, "Exploiting on-device image classification for energy efficiency in ambient-aware systems," in *Mobile Cloud Visual Media Computing*, pp. 167–199, Springer, 2015.

[29] X. Ran, H. Chen, X. Zhu, Z. Liu, and J. Chen, "DeepDecision: A mobile deep learning framework for edge video analytics," in *Proceedings of IEEE Conference on Computer Communications (INFOCOM)*, pp. 1421–1429, 2018.

[30] Q. Liu, S. Huang, J. Opadere, and T. Han, "An edge network orchestrator for mobile augmented reality," in *Proceedings of IEEE Conference on Computer Communications (INFOCOM)*, pp. 756–764, 2018.

[31] A. Khalili, S. Zarandi, and M. Rasti, "Joint resource allocation and offloading decision in mobile edge computing," *IEEE Communications Letters*, vol. 23, no. 4, pp. 684–687, 2019.

[32] J. Wang, J. Tang, Z. Xu, Y. Wang, G. Xue, X. Zhang, and D. Yang, "Spatiotemporal modeling and prediction in cellular networks: A big data enabled deep learning approach," in *IEEE International Conference on Computer Communications (INFOCOM)*, (Atlanta, GA, USA), May 2017.

[33] K. Zheng, Z. Yang, K. Zhang, P. Chatzimisios, K. Yang, and W. Xiang, "Big data-driven optimization for mobile networks toward 5G," *IEEE Network*, vol. 30, no. 1, pp. 44–51, 2016.

[34] M. K. Weldon, *The Future X Network: A Bell Labs Perspective*. Crc Press, 2016.

[35] X. Ge, S. Tu, G. Mao, C. X. Wang, and T. Han, "5G ultra-dense cellular networks," *IEEE Wireless Comm.*, vol. 23, no. 1, pp. 72–79, 2016.

[36] Q. Ye, B. Rong, Y. Chen, M. Al-Shalash, C. Caramanis, and J. G. Andrews, "User association for load balancing in heterogeneous cellular networks," *IEEE Transactions on Wireless Communications*, vol. 12, no. 6, pp. 2706–2716, 2013.

[37] J. Wu, Y. Zhang, M. Zukerman, and E. K.-N. Yung, "Energy-efficient base-stations sleep-mode techniques in green cellular networks: A survey," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 2, pp. 803–826, 2015.

[38] S. Huang, T. Han, and N. Ansari, "Big-data-driven network partitioning for ultra-dense radio access networks," in *IEEE International Conference on Communications (ICC)*, (Paris, France), May 2017.

[39] J. G. Andrews, S. Singh, Q. Ye, X. Lin, and H. S. Dhillon, "An overview of load balancing in HetNets: Old myths and open problems," *IEEE Wireless Communications*, vol. 21, no. 2, pp. 18–25, 2014.

[40] T. Han and N. Ansari, "A traffic load balancing framework for software-defined radio access networks powered by hybrid energy sources," *IEEE/ACM Transactions on Networking*, vol. 24, pp. 1038–1051, April 2016.

[41] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 236–262, 2016.

[42] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network function virtualization: Challenges and opportunities for innovations," *IEEE Communications Magazine*, vol. 53, no. 2, pp. 90–97, 2015.

[43] V. Eramo, E. Miucci, M. Ammar, and F. G. Lavacca, "An approach for service function chain routing and virtual function network instance migration in network function virtualization architectures," *IEEE/ACM Transactions on Networking*, 2017.

[44] S. C. Johnson, "Hierarchical clustering schemes," *Psychometrika*, vol. 32, no. 3, pp. 241–254, 1967.

[45] Y. Sun, X. Yin, J. Jiang, V. Sekar, F. Lin, N. Wang, T. Liu, and B. Sinopoli, "CS2P: Improving video bitrate selection and adaptation with data-driven throughput prediction," in *Proceedings of the ACM SIGCOMM Conference*, pp. 272–285, 2016.

[46] Z. Li, A. Aaron, I. Katsavounidis, A. Moorthy, and M. Manohara, "Toward a practical perceptual video quality metric," *The Netflix Tech Blog*, vol. 6, 2016.

[47] "Video codec—H.264 standardization." `https://www.itu.int/rec/T-REC-H.264`.

[48] "Video codec—VP9." `https://www.webmproject.org/vp9/`.

[49] Y. Zhang, Y. Zhang, Y. Wu, Y. Tao, K. Bian, P. Zhou, L. Song, and H. Tuo, "Improving quality of experience by adaptive video streaming with super-resolution," in *Proceedings of IEEE Conference on Computer Communications (INFOCOM)*, pp. 1957–1966, 2020.

[50] C. Dong, C. C. Loy, and X. Tang, "Accelerating the super-resolution convolutional neural network," in *Proceedings of the European Conference on Computer Vision*, pp. 391–407, 2016.

[51] B. Lim, S. Son, H. Kim, S. Nah, and K. M. Lee, "Enhanced deep residual networks for single image super-resolution," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, July 2017.

[52] X. Wang, K. Yu, S. Wu, J. Gu, Y. Liu, C. Dong, Y. Qiao, and C. C. Loy, "ESRGAN: Enhanced super-resolution generative adversarial networks," in *Proceedings of the European Conference on Computer Vision Workshops (ECCVW)*, September 2018.

[53] J.-H. Heu, D.-Y. Hyun, C.-S. Kim, and S.-U. Lee, "Image and video colorization based on prioritized source propagation," in *Proceedings of the IEEE International Conference on Image Processing (ICIP)*, pp. 465–468, 2009.

[54] A. Levin, D. Lischinski, and Y. Weiss, "Colorization using optimization," *ACM Transactions on Graphics (TOG)*, vol. 23, no. 3, pp. 689–694, 2004.

[55] R. Zhang, J.-Y. Zhu, P. Isola, X. Geng, A. S. Lin, T. Yu, and A. A. Efros, "Real-time user-guided image colorization with learned deep priors," *ACM Transactions on Graphics (TOG)*, vol. 9, no. 4, 2017.

[56] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1125–1134, 2017.

[57] O. Rippel and L. Bourdev, "Real-time adaptive image compression," in *Proceedings of the 34th International Conference on Machine Learning*, pp. 2922–2930, 2017.

[58] E. Agustsson, F. Mentzer, M. Tschannen, L. Cavigelli, R. Timofte, L. Benini, and L. V. Gool, "Soft-to-hard vector quantization for end-to-end learning compressible representations," in *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, pp. 1141–1151, 2017.

[59] M. Tschannen, E. Agustsson, and M. Lucic, "Deep generative models for distribution-preserving lossy compression," in *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, pp. 5929–5940, 2018.

[60] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman, "BOLA: Near-optimal bitrate adaptation for online videos," in *Proceedings of IEEE International Conference on Computer Communications*, pp. 1–9, 2016.

[61] J. Joskowicz and J. C. L. Ardao, "Combining the effects of frame rate, bit rate, display size and video content in a parametric video quality model," in *Proceedings of the 6th Latin America Networking Conference*, pp. 4–11, 2011.

[62] G. Zhai, J. Cai, W. Lin, X. Yang, and W. Zhang, "Three dimensional scalable video adaptation via user-end perceptual quality assessment," *IEEE Transactions on Broadcasting*, vol. 54, no. 3, pp. 719–727, 2008.

[63] L. Yue, H. Shen, J. Li, Q. Yuan, H. Zhang, and L. Zhang, "Image super-resolution: The techniques, applications, and future," *Signal Processing*, vol. 128, pp. 389–408, 2016.

[64] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[65] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," *Computer vision–ECCV 2006*, pp. 404–417, 2006.

[66] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *Computer Vision (ICCV), 2011 IEEE international conference on*, pp. 2564–2571, IEEE, 2011.

[67] J. Redmon and A. Farhadi, "Yolo9000: Better, faster, stronger," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[68] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*, pp. 21–37, Springer, 2016.

[69] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, pp. 91–99, 2015.

[70] 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," *arXiv preprint arXiv:1704.04861*, 2017.

[71] M. Apte, S. Mangat, and P. Sekhar, "Yolo net on ios," 2017.

[72] L. Liu, H. Li, and M. Gruteser, "Edge assisted real-time object detection for mobile augmented reality," in *Proceedings of ACM Conference on Mobile Computing and Networking (MobiCom)*, pp. 1–16, 2019.

[73] K. Kumar, J. Liu, Y.-H. Lu, and B. Bhargava, "A survey of computation offloading for mobile systems," *Mobile Networks and Applications*, vol. 18, no. 1, pp. 129–140, 2013.

[74] T. Q. Dinh, J. Tang, Q. D. La, and T. Q. Quek, "Offloading in mobile edge computing: Task allocation and computational frequency scaling," *IEEE Transactions on Communications*, vol. 65, no. 8, pp. 3571–3584, 2017.

[75] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.

[76] N. Inoue, R. Furuta, T. Yamasaki, and K. Aizawa, "Cross-domain weakly-supervised object detection through progressive domain adaptation," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5001–5009, 2018.

[77] D. Li, S. Tasci, S. Ghosh, J. Zhu, J. Zhang, and L. Heck, "RILOD: near real-time incremental learning for object detection at the edge," in *Proceedings of ACM/IEEE Symposium on Edge Computing (SEC)*, pp. 113–126, 2019.

[78] M. Xia, M. Shirazipour, Y. Zhang, H. Green, and A. Takacs, "Network function placement for nfv chaining in packet/optical datacenters," *Journal of Lightwave Technology*, vol. 33, no. 8, pp. 1565–1570, 2015.

[79] M. Bouet, J. Leguay, T. Combe, and V. Conan, "Cost-based placement of vdpi functions in nfv infrastructures," *International Journal of Network Management*, vol. 25, no. 6, pp. 490–506, 2015.

[80] S. Mehraghdam, M. Keller, and H. Karl, "Specifying and placing chains of virtual network functions," in *IEEE International Conference on Cloud Networking (CloudNet)*, pp. 7–13, IEEE, 2014.

[81] M. Ghaznavi, A. Khan, N. Shahriar, K. Alsubhi, R. Ahmed, and R. Boutaba, "Elastic virtual network function placement," in *Cloud Networking (CloudNet), 2015 IEEE 4th International Conference on*, pp. 255–260, IEEE, 2015.

[82] Q. Zhang, Y. Xiao, F. Liu, J. C. Lui, J. Guo, and T. Wang, "Joint optimization of chain placement and request scheduling for network function virtualization," in *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*, pp. 731–741, IEEE, 2017.

[83] R. Cziva, C. Anagnostopoulos, and D. P. Pezaros, "Dynamic, latency-optimal vnf placement at the network edge," 2018.

[84] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, "Near optimal placement of virtual network functions," in *Computer Communications (INFOCOM), 2015 IEEE Conference on*, pp. 1346–1354, IEEE, 2015.

[85] X. Zhang, C. Wu, Z. Li, and F. C. Lau, "Proactive vnf provisioning with multi-timescale cloud resources: Fusing online learning and online optimization," in *INFOCOM 2017-IEEE Conference on Computer Communications, IEEE*, pp. 1–9, IEEE, 2017.

[86] S. Q. Zhang, A. Tizghadam, B. Park, H. Bannazadeh, and A. Leon-Garcia, "Joint nfv placement and routing for multicast service on sdn," in *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, pp. 333–341, IEEE, 2016.

[87] M. C. Luizelli, L. R. Bays, L. S. Buriol, M. P. Barcellos, and L. P. Gaspary, "Piecing together the nfv provisioning puzzle: Efficient placement and chaining of virtual network functions," in *IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pp. 98–106, IEEE.

[88] F. Schardong, I. Nunes, and A. Schaeffer-Filho, "A distributed nfv orchestrator based on bdi reasoning," in *IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pp. 107–115, IEEE, 2017.

[89] M. C. Luizelli, D. Raz, Y. Sa'ar, and J. Yallouz, "The actual cost of software switching for nfv chaining," in *IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pp. 335–343, IEEE, 2017.

[90] M. F. Bari, S. R. Chowdhury, R. Ahmed, and R. Boutaba, "On orchestrating virtual network functions," in *International Conference on Network and Service Management (CNSM)*, pp. 50–56, IEEE, 2015.

[91] Y. Li and M. Chen, "Software-defined network function virtualization: A survey," *IEEE Access*, vol. 3, pp. 2542–2553, 2015.

[92] M. C. Luizelli, L. R. Bays, L. S. Buriol, M. P. Barcellos, and L. P. Gaspary, "Piecing together the nfv provisioning puzzle: Efficient placement and chaining of virtual network functions," in *IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pp. 98–106, IEEE, 2015.

[93] J. Liu, F. Liu, and N. Ansari, "Monitoring and analyzing big traffic data of a large-scale cellular network with hadoop," *IEEE Network*, vol. 28, no. 4, pp. 32–39, 2014.

[94] F. Xu, Y. Li, M. Chen, and S. Chen, "Mobile cellular big data: linking cyberspace and the physical world with social ecology," *IEEE network*, vol. 30, no. 3, pp. 6–12, 2016.

[95] Z. Su, Q. Xu, and Q. Qi, "Big data in mobile social networks: A QoE-oriented framework," *IEEE Network*, vol. 30, no. 1, pp. 52–57, 2016.

[96] C. R. Lin and M. Gerla, "Adaptive clustering for mobile wireless networks," *IEEE Journal on Selected areas in Communications*, vol. 15, no. 7, pp. 1265–1275, 1997.

[97] J. Y. Yu and P. H. J. Chong, "A survey of clustering schemes for mobile ad hoc networks," *IEEE Communications Surveys & Tutorials*, vol. 7, no. 1, pp. 32–48, 2005.

[98] S. Banerjee and S. Khuller, "A clustering scheme for hierarchical control in multi-hop wireless networks," in *INFOCOM 2001. Twentieth annual joint conference of the IEEE computer and communications societies. Proceedings. IEEE*, vol. 2, pp. 1028–1037, IEEE, 2001.

[99] K. H. Wang and B. Li, "Group mobility and partition prediction in wireless ad-hoc networks," in *Communications, 2002. ICC 2002. IEEE International Conference on*, vol. 2, pp. 1017–1021, IEEE, 2002.

[100] Y. Yu, S. Murphy, and L. Murphy, "A clustering approach to planning base station and relay station locations in ieee 802.16 j multi-hop relay networks," in *Communications, 2008. ICC'08. IEEE International Conference on*, pp. 2586–2591, IEEE, 2008.

[101] M. Hong, R. Sun, H. Baligh, and Z.-Q. Luo, "Joint base station clustering and beamformer design for partial coordinated transmission in heterogeneous networks," *IEEE Journal on Selected Areas in Communications*, vol. 31, no. 2, pp. 226–240, 2013.

[102] D. Ray, J. Kosaian, K. Rashmi, and S. Seshan, "Vantage: optimizing video upload for time-shifted viewing of social live streams," in *Proceedings of ACM Special Interest Group on Data Communication (SIGCOMM)*, pp. 380–393, 2019.

[103] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli, "A control-theoretic approach for dynamic adaptive video streaming over HTTP," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 325–338, 2015.

[104] G. Yi, D. Yang, A. Bentaleb, W. Li, Y. Li, K. Zheng, J. Liu, W. T. Ooi, and Y. Cui, "The ACM Multimedia 2019 Live Video Streaming Grand Challenge," in *Proceedings of ACM International Conference on Multimedia*, pp. 2622–2626, 2019.

[105] 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 ACM SIGCOMM Conference*, vol. 44, pp. 187–198, 2014.

[106] F. Y. Yan, H. Ayers, C. Zhu, S. Fouladi, J. Hong, K. Zhang, P. Levis, and K. Winstein, "Learning in situ: a randomized experiment in video streaming," in *Proceedings of USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pp. 495–511, 2020.

[107] Y.-F. Ou, Z. Ma, T. Liu, and Y. Wang, "Perceptual quality assessment of video considering both frame rate and quantization artifacts," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 21, no. 3, pp. 286–298, 2010.

[108] C. J. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[109] "A virtual network simulator: Mininet." `http://mininet.org/`.

[110] J. Liu, M. Lu, K. Chen, X. Li, S. Wang, Z. Wang, E. Wu, Y. Chen, C. Zhang, and M. Wu, "Overfitting the data: Compact neural video delivery via content-aware feature modulation," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 4631–4640, 2021.

[111] R. Lee, S. I. Venieris, and N. D. Lane, "Neural enhancement in content delivery systems: The state-of-the-art and future directions," in *Proceedings of the 1st Workshop on Distributed Machine Learning*, pp. 34–41, 2020.

[112] Y. Wu and Y. Tian, "Training agent for first-person shooter game with actor-critic curriculum learning," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

[113] J. van der Hooft, S. Petrangeli, M. Claeys, J. Famaey, and F. De Turck, "A learning-based algorithm for improved bandwidth-awareness of adaptive streaming clients," in *Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pp. 131–138, 2015.

[114] T.-Y. Huang, N. Handigol, B. Heller, N. McKeown, and R. Johari, "Confused, timid, and unstable: picking a video streaming rate is hard," in *Proceedings of the ACM Internet Measurement Conference*, pp. 225–238, 2012.

[115] T.-C. Wang, M.-Y. Liu, J.-Y. Zhu, A. Tao, J. Kautz, and B. Catanzaro, "High-resolution image synthesis and semantic manipulation with conditional GANs," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[116] M. He, D. Chen, J. Liao, P. V. Sander, and L. Yuan, "Deep exemplar-based colorization," *ACM Transactions on Graphics (TOG)*, vol. 37, no. 4, p. 47, 2018.

[117] Y. Luo and X. Tang, "Photo and video quality evaluation: Focusing on the subject," in *Proceedings of the European Conference on Computer Vision*, pp. 386–399, Springer, 2008.

[118] E. Agustsson and R. Timofte, "NTIRE 2017 Challenge on Single Image Super-Resolution: Dataset and Study," in *Processings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, July 2017.

[119] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the ACM International Conference on Multimedia*, pp. 675–678, 2014.

[120] "ARCore." `https://developers.google.com/ar/`.

[121] "ARkit." `https://developer.apple.com/arkit/`.

[122] D. Chatzopoulos, C. Bermejo, Z. Huang, and P. Hui, "Mobile augmented reality survey: From where we are to where we go," *IEEE Access*, vol. 5, pp. 6917–6950, 2017.

[123] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *Proceedings of European Conference on Computer Vision (ECCV)*, pp. 740–755, 2014.

[124] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, *et al.*, "Speed/accuracy trade-offs for modern convolutional object detectors," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7310–7311, 2017.

[125] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and F.-F. Li, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[126] A. Kuznetsova, H. Rom, N. Alldrin, J. Uijlings, I. Krasin, J. Pont-Tuset, S. Kamali, S. Popov, M. Malloci, T. Duerig, *et al.*, "The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale," *arXiv preprint arXiv:1811.00982*, 2018.

[127] H. Wang and J. Xie, "User preference based energy-aware mobile AR system with edge computing," in *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM)*, pp. 1379–1388, 2020.

[128] B. Liu, Y. Li, Y. Liu, Y. Guo, and X. Chen, "Pmc: A privacy-preserving deep learning model customization framework for edge computing," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 4, no. 4, pp. 1–25, 2020.

[129] S. Huang, T. Han, and J. Xie, "A smart-decision system for realtime mobile augmented reality applications," in *Proceedings of IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, 2019.

[130] 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 *Proceedings of ACM Conference on Embedded Networked Sensor Systems (SenSys)*, pp. 96–109, 2019.

[131] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[132] ITU Recommendation and P.910, "Subjective video quality assessment methods for multimedia applications," 2008.

[133] Y. Qin, S. Hao, K. R. Pattipati, F. Qian, S. Sen, B. Wang, and C. Yue, "ABR streaming of VBR-encoded Videos: Characterization, challenges, and solutions," in *Proceedings of ACM Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, pp. 366–378, 2018.

[134] Y. Bao, H. Wu, A. A. Ramli, B. Wang, and X. Liu, "Viewing 360-degree videos: Motion prediction and bandwidth optimization," in *Proceedings of IEEE International Conference on Network Protocols (ICNP)*, pp. 1–2, 2016.

[135] Y. Bao, T. Zhang, A. Pande, H. Wu, and X. Liu, "Motion-prediction-based multicast for 360-degree video transmissions," in *Proceedings of IEEE International Conference on Sensing, Communication, and Networking (SECON)*, pp. 1–9, 2017.

[136] C. J. Willmott and K. Matsuura, "Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance," *Climate Research*, vol. 30, no. 1, pp. 79–82, 2005.

[137] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 248–255, IEEE, 2009.

[138] N. Ansari and E. Hou, *Computational intelligence for optimization*. Springer Science & Business Media, 2012.

[139] L. Kaufman and P. Rousseeuw, *Clustering by means of medoids*. North-Holland, 1987.

[140] H.-S. Park and C.-H. Jun, "A simple and fast algorithm for k-medoids clustering," *Expert systems with applications*, vol. 36, no. 2, pp. 3336–3341, 2009.

[141] W. Gellert, S. Gottwald, M. Hellwich, H. Kästner, and H. Küstner, "Spherical trigonometry," in *The VNR Concise Encyclopedia of Mathematics*, pp. 261–282, Springer, 1975.

[142] H. Alt and M. Codau, "Computing the Fréchect distance between two polygonal curves," *International Journal of Computational Geometry and Applications*, vol. 05, no. 01n02, pp. 75–91, 1995.

[143] P. Geladi and B. R. Kowalski, "Partial least-squares regression: a tutorial," *Analytica chimica acta*, vol. 185, pp. 1–17, 1986.

[144] V. Cherkassky and Y. Ma, "Practical selection of svm parameters and noise estimation for svm regression," *Neural networks*, vol. 17, no. 1, pp. 113–126, 2004.

[145] G. Zhang, B. E. Patuwo, and M. Y. Hu, "Forecasting with artificial neural networks:: The state of the art," *International journal of forecasting*, vol. 14, no. 1, pp. 35–62, 1998.

[146] W.-C. Hong, "Application of seasonal svr with chaotic immune algorithm in traffic flow forecasting," *Neural Computing and Applications*, vol. 21, no. 3, pp. 583–593, 2012.

[147] T. Srinivasan, V. Vijaykumar, and R. Chandrasekar, "A self-organized agent-based architecture for power-aware intrusion detection in wireless ad-hoc networks," in *IEEE International Conference on Computing and Informatics (ICOCI)*, (Kuala Lumpur, Malaysia), June 2006.

[148] U. Rupapara and G. Mulchandani, "Cancer diagnosis using clustering technique: A literature survey," *Data Mining and Knowledge Engineering*, vol. 8, no. 2, pp. 44–47, 2016.