Learning-based Adaptive Video Streaming

by

Xiaolan Jiang

Dissertation

submitted to the Department of Informatics
in partial fulfilment of the requirements for the degree of

Doctor of Philosophy

The Graduate University for Advanced Studies, SOKENDAI

September 2020

Abstract

In recent years, video streaming traffic has been increasing significantly. To satisfy and attract more users, video applications not only struggle with delivering high-quality videos over the Internet, but also develop new types of video streaming scenarios. Recent advances of hardware and network technology have made two new video streaming scenarios, live and 360-degree video streaming, become tremendously popular among users. However, new streaming scenarios are making it more challenging to provide higher user perceived quality of experience (QoE).

In live streaming scenarios, high video quality and low latency are two main requirements. Unlike on-demand video streaming services where only bitrate decisions need to be made by the applications, a live streaming application needs to make more complicated options (e.g. target buffer level, latency limit) to achieve low latency. Moreover, future video chunks' information are unknown in live streaming services, and it leads to a more difficult task to make decisions.

Challenges for streaming high-quality 360-degree videos mainly comes from its excessive need of network bandwidth. To prevent the delivery of entire 360 videos from adversely affecting QoE, tile-based viewport adaptive streaming that divides 360 video chunks into tiles and conveys streams with differentiated quality levels to viewport and non-viewport areas has been regarded as a promising solution. Existing works have been devoted to the design of viewport prediction (VPP) to predict users' viewport orientation due to head movements, as well as tile bitrate selection (TBS) to determine tile-based bitrates for viewport and non-viewport areas. These two parts are still an open challenge for the research community due to unpredictable user head rotation patterns and large

action space.

Our key insight is that learning-based methods can see human-invisible network and user patterns well through a large amount data traces. This allows us to apply learning-based schemes to tackle the challenges in live and 360-degree video streaming scenarios. We formulate the live streaming task as a reinforcement learning problem with discrete-continuous hybrid action spaces, then propose a novel deep reinforcement learning (DRL) algorithm to train an neural agent which can take hybrid actions.

For 360-degree video streaming, we use part of non-viewport areas to help resist prediction errors, and divide all tiles into three areas: viewport, adjacent and out areas. Then we train a TBS agent with DRL to determine bitrate for each tile area. To predict future viewport more accurately, we propose a sinusoidal viewport prediction system which leverages the sinusoidal values of rotation angles for orientation prediction on *yaw* direction, and utilize the correlation between predictions errors and head movement velocity as well as prediction time window to handle the potential prediction error.

To evaluate the performance of the proposed schemes we conduct extensive simulations based on various categories of videos, and real-world bandwidth and user head motion datasets. Simulation results demonstrate that the proposed schemes achieve better QoE than comparison schemes.

The key contribution of this dissertation is to improve user-perceived QoE by learning-based approaches in both live and 360-degree video streaming scenarios. More specifically, in live video streaming scenario, we design a novel DRL algorithm to take hybrid actions which outperforms many standard DRL algorithms. For 360-degree video streaming, we find that utilizing sinusoidal values of rotation angles as features can significantly reduce viewport prediction error and design a system based on this finding, and we apply DRL to train a neural TBS agent which achieves better performance than heuristic-based methods.

Acknowledgement

First and foremost, I would like to express a lot gratitude to my supervisor Prof. Yusheng Ji. During my five-year research career, she has acted as a tireless mentor, counselor, and co-author. Prof. Ji has offered me many insightful advises to my study, and every meeting I recall with her ended being inspired and encouraged. I was deeply indebted to her for allowing and supporting me change my research topic from network caching to learning-based video streaming, and this will have a significant influence in my future career life. Besides research, she has shared her wisdom and experiences in life and career to me. She is a nice, generous, and patient person in my eyes, and my personality has been inevitably by her throughout the years. I am extremely lucky to have her as my supervisor in the past five years.

I am very thankful to my sub-advisors, Prof. Kensuke Fukuda and Prof. Shunji Abe, and other members of my committee, Prof. Megumi Kaneko, Prof. Takashi Kurimoto, and Prof. Jiro Katto. I am grateful to my co-authors, Dr. Yi-Han Chiang, Dr. Zhi Liu, Mr. Si Ahmed NAAS, Prof. Ying Cui, Dr. Lei Zhong, and Prof. Stephan Sigg, as well as other members in Prof. Ji's lab, for their helpful suggestions and in-depth discussions. Yi-Han was a postdoctoral researcher in NII and now an assistant professor at Osaka Prefecture University. He is good at theorem proving and writing, and he has given me great help on proofreading my research paper. At the early years in my research, Zhi has given me many suggestions to my research topic and provided me great help and guidance in writing a research paper. Ahmed contributed some implementation to part of my dissertation. Ying, Lei, and Stephan has given me comments on improving the quality of my research paper. Without them, I would have had a harder research life.

Finally, I would like to express my earnest gratitude to my parents for their constant support and love. Without them, all of my current achievements would have been impossible.

Contents

Abstract iii

Acknowledgement v

List of Figures 1

1 Introduction 1

1.1 Background for Adaptive Video Streaming . . . 2

1.1.1 Live Video Streaming . . . 3

1.1.2 360-degree Video Streaming . . . 4

1.2 QoE Metrics for Video Streaming . . . 6

1.2.1 Subjective QoE Metric. . . . 7

1.2.2 Objective QoE Metric. . . . 7

1.3 Challenges . . . 9

1.3.1 Universal Challenges for Video Streaming . . . 10

1.3.2 New Challenges for Live Video Streaming . . . 10

1.3.3 New Challenges for 360-degree Video Streaming . . . 11

1.4 Learning as a Solution for Video Streaming . . . 11

1.5 Organization . . . 14

2 Related Work 15

2.1 Live Video Streaming . . . 15

2.1.1 ABR algorithms for VoD services . . . 15

2.1.2 ABR algorithms for Live streaming . . . 16
2.2 360-degree Video Streaming . . . 19
2.3 Viewport Prediction . . . 20

3 Learning-based Live Video Streaming 25
3.1 Overview . . . 25
3.2 System Design . . . 26
3.2.1 Reinforcement Learning for Live Streaming . . . 26
3.2.2 HD2: Dueling DQN with Hybrid Action Space . . . 29
3.2.3 HD3: Distributed HD2 . . . 30
3.3 Evaluation . . . 32
3.3.1 QoE Definition . . . 32
3.3.2 Dataset Analysis . . . 33
3.3.3 Comparison Schemes . . . 35
3.3.4 Implementation . . . 35
3.3.5 Results Analysis . . . 36
3.4 Summary . . . 40

4 Learning-based 360-Degree Video Streaming 41
4.1 Overview . . . 41
4.2 System Design . . . 42
4.2.1 VPP – The Prediction of Viewport . . . 43
4.2.2 TBS – The Bitrate Selection of Tiles . . . 45
4.3 Evaluation . . . 49
4.3.1 Parameter Settings . . . 49
4.3.2 Datasets . . . 50
4.3.3 360 Video Player Simulator Design . . . 52
4.3.4 Comparison Metrics . . . 53
4.3.5 QoE Metrics . . . 54
4.3.6 Reward function design . . . 55
4.3.7 Result Analysis . . . 56

4.4 Summary . . . 57

5 Sinusoidal Viewport Prediction for 360-Degree Video Streaming 59
5.1 Overview . . . 59
5.2 Sinusoids versus Prediction Accuracy . . . 63
5.2.1 Prediction Error Analysis . . . 63
5.2.2 Conversion of Degrees to Sinusoid . . . 63
5.2.3 Why sinusoid is better? . . . 64
5.3 System Design . . . 66
5.3.1 Orientation Prediction . . . 66
5.3.2 Error Handling . . . 67
5.3.3 Tile Probability Normalization . . . 68
5.4 Evaluation . . . 71
5.4.1 Simulation Settings . . . 71
5.4.2 Viewport Prediction Accuracy . . . 73
5.4.3 Video Quality Assessment . . . 74
5.5 Summary . . . 84

6 Conclusion 85
6.1 Contributions . . . 85
6.2 Discussion . . . 86
6.2.1 Limitations . . . 86
6.2.2 Lessons Learned . . . 88
6.3 Future Perspective . . . 89
6.3.1 New networking challenges solvable by learning . . . 89
6.3.2 Integrating heuristics with reinforcement learning . . . 90

Bibliography 93

List of Figures

1.1 An overview of HTTP adaptive video streaming. 3

1.2 The mapping of a spherical surface into a plane. 5

1.3 The main contribution of this dissertation is to develop three learning-based solutions (bottom) to address the key challenges (middle) exposed by live and 360-degree video streaming scenarios (top). 14

3.1 The Workflows of RL. 27

3.2 HD2 Architecture 28

3.3 CDF of Bandwidth in Four Network Conditions. 33

3.4 GOP sizes of different bitrates from the same video 34

3.5 Ratios of GOP sizes between adjacent bitrates from the same video . . . 34

3.6 Comparing HD3 with several state of the art DRL algorithms on four kinds of network environments (high, medium, low, oscillating) in live streaming scenarios. Results are normalized against the performance of HD3. 36

3.7 Analyzing performance of different DRL algorithms on different granularity of discretization of latency limit value on *High* bandwidth scenario. Results are normalized against the performance of HD3 which outputs a continuous latency limit value. 38

3.8 Analyzing performance of different DRL algorithms on different granularity of discretization of latency limit value on *Medium* bandwidth scenario. Results are normalized against the performance of HD3 which outputs a continuous latency limit value. . . . 38
3.9 Analyzing performance of different DRL algorithms on different granularity of discretization of latency limit value on *Low* bandwidth scenario. Results are normalized against the performance of HD3 which outputs a continuous latency limit value. . . . 39
3.10 Analyzing performance of different DRL algorithms on different granularity of discretization of latency limit value on *Oscillating* bandwidth scenario. Results are normalized against the performance of HD3 which outputs a continuous latency limit value. . . . 39

4.1 The architecture of Plato. . . . 43
4.2 LSTM based predictor. . . . 44
4.3 Division of tile areas. . . . 45
4.4 The workflows of RL. . . . 46
4.5 An illustration of A3C algorithm. . . . 48
4.6 CDF of bandwidth 51
4.7 sample of 4G bandwidth 53
4.8 Average QoE 56
4.9 Comparing Plato with existing algorithms by analyzing their performance on the individual components in the general QoE definition 57

5.1 An illustration of tile-based streaming for 360-degree videos in VoD and live scenarios. . . . 61
5.2 The prediction errors with respect to various time window lengths on the AV dataset (see Sec. 5.4.1). . . . 62
5.3 Head movement trace on *yaw* direction. The angle is represented by degree, and cosine/sine of the angle. . . . 64
5.4 An illustration of predicted values of models trained with both the degrees and the sinusoidal values. . . . 64

5.5 An illustration of the SVP system, which consists of three stages: orientation prediction, error handling, and tile probability normalization. The SVP system predicts based on head rotations collected by an HMD, and finally outputs viewing probability for each tile to the ABR model (corresponding to Figure 5.1). 69
5.6 The relationship between prediction errors, time window lengths and head movement velocities in the AV dataset. 70
5.7 The prediction errors on *yaw* direction in 1-, 3-, 5-sec time window. . . . 77
5.8 The prediction errors on *pitch* direction in 1-, 3-, 5-sec time window. . . . 78
5.9 Tile prediction accuracy on different datasets. 79
5.10 Effective bitrate comparison (fixed bandwidth, 3-sec buffer threshold). . . 80
5.11 Effective bitrate comparison (dynamic bandwidth, 3-sec buffer threshold). 81
5.12 Effective bitrate comparison (dynamic bandwidth, 6-sec buffer threshold). 82
5.13 Effective bitrate improvement by each stage of SVP (dynamic bandwidth, 3- and 6-sec buffer thresholds). 83

List of Abbreviations

A3C	Asynchronous Advantage Actor Critic
ABR	Adaptive Bitrate
CDN	Content Delivery Network
CNN	Convolutional Neural Network
DASH	Dynamic Adaptive Streaming over HTTP
DQN	Deep Q Network
DRL	Deep Reinforcement Learning
FC	Fully Connected
FoV	Field of View
GOP	Group of Pictures
HD3	Hybrid Distributed Dueling DQN
HMD	Head Mounted Displays
HTTP	Hypertext Transfer Protocol
LinSVR	Linear Support Vector Regression
LR	Linear Regression

LSTM	Long Short Term Memory
MOS	Mean Opinion Score
MPC	Model Predictive Control
MPD	Media Presentation Description
PSNR	Peak Signal-to-Noise Ratio
PSPNR	Peak Perceptible-Noise Ratio
QoE	Quality of Experience
SSIM	Structural Similarity Index
SVP	Sinusoidal Viewport Prediction
TBS	Tile Bitrate Selection
VMAF	Video Multimethod Assessment Fusion
VoD	Video on Demand
VPP	Viewport Prediction
VR	Virtual Reality

1 Introduction

As the development of hardware technology, more and more people are holding one or several smart devices which are accessible to the Internet. To attract users and increase the revenues, many applications have been developed to provide services to the Internet users for various purposes (e.g., video, gaming, file sharing, etc). The ever-increasing numbers of Internet users and applications have triggered the explosive growth of the global IP traffic. By 2022, global IP traffic will increase to nearly 11 times more than all IP traffic generated in 2012, reaching an annual rate of 4.8 zettabytes per year. Among numerous Internet applications, video streaming has contributed the largest share of the global Internet traffic. The video traffic is expected to account for 82% of the global Internet traffic by 2022, up from 73% in 2017. Furthermore, live videos will increase 15-fold and reach about 17% of the total video traffic, and virtual reality (VR) traffic will increase 12-fold by 2022 [1].

The huge demand for Internet videos (short, long, 360-degree, live videos) has made

video streaming a popular yet competitive service, content providers need to provide better quality of experience (QoE) to attract more users to increase their revenues. One of the latest definitions of QoE is [2]:

> *QoE is the degree of delight or annoyance of the user of an application or service. It results from the fulfillment of his or her expectations with respect to the utility and/or enjoyment of the application or service in the light of the user's personality and current state.*

However, it's non-trival to design video streaming systems to achieve high QoE due to various challenges caused by dynamic network conditions and specific requirements from different video streaming scenarios. Users may quickly quit a video session if the video quality is insufficient or there are frequent rebufferring events. Dynamic network conditions and limited bandwidth prevent content providers from delivering high-quality videos to the users.

In the following, we'll first introduce the background for adaptive video streaming scenarios and talk about the corresponding challenges. Then we give a brief introduction to our proposed learning-based solutions to tackle the challenges of live and 360-degree video streaming scenarios.

1.1 Background for Adaptive Video Streaming

To overcome the dynamic network issue, many video content providers have deployed HTTP-based adaptive streaming (standardized as DASH [3]) system in their applications. In DASH systems, a video is temporally partitioned into multiple segments, each of which is then encoded at several bitrate levels, where a higher bitrate represents a higher video quality and thus larger segment size. During video streaming, a video player can adaptively select an appropriate bitrate for each segment based on the estimate of the future network bandwidth. Video segments are stored on HTTP servers along with a manifest file (i.e. MPD file) describing video's information.

A typical end-to-end video-on-demand (VoD) streaming process is shown in Figure 1.1. At the beginning, a video player embedded in a client application first requests the

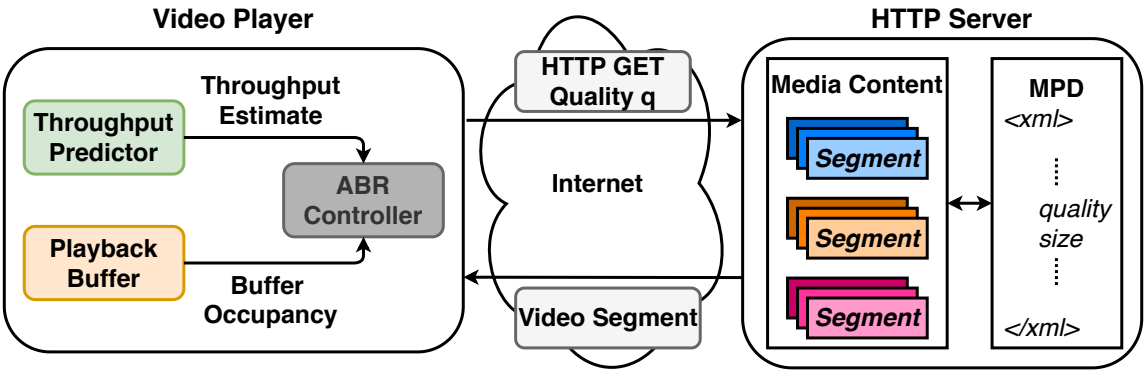


Figure 1.1: An overview of HTTP adaptive video streaming.

MPD file from the HTTP server. Then at each adaptation step, the player requests next video segment at quality q according to an adaptive bitrate (ABR) algorithm. These algorithms take various inputs (e.g., throughput estimation, playback buffer level, etc.) to make the bitrate decision for next segment. As segments are downloaded, they will be stored in the buffer and then played back to the client. It is noted that playback of a given segment cannot start until the downloading process for the entire segment has been finished.

With the development of technology and network, two new video scenarios: 1) live video streaming and 2) 360-degree video streaming are becoming popular nowadays. In the following, we'll give an overview of these two promising scenarios.

1.1.1 Live Video Streaming

In recent years, live video streaming has become tremendously popular [4, 5]. Many video applications (e.g., Twitch, Facebook Live, Kwai, Douyu) has been developed to allow users to broadcast real-time videos over the Internet, and interact with their viewers. To satisfy the users, live video streaming services have to meet two requirements: high quality of videos and low latency due to the real-time interaction between the broadcasters and viewers.

A typical live video streaming scenario can be considered as follows. There is a user

captures and produces a video in real-time at anywhere with any device (mobile phone or PC). The lively recorded video stream is uploaded to the server where the same video will be transcoded into multiple quality or bitrate levels. All quality levels of videos are then delivered to CDN (content delivery network) nodes, which are located at the edge of the network and near to the end users. The viewers send requests to the CDN nodes to prefetch one specific quality level of video based on their own network conditions.

Unlike VoD streaming services where only bitrate decisions need to be made by the clients, there are more options for the client in a live streaming scenario described as above. Specifically in this live streaming grand challenge [6], a client needs to make three decisions of 1) deciding which bitrate to prefetch to increase the video quality while avoiding rebuffering and bitrate fluctuating, given its own network condition, 2) selecting target buffer level to decide whether slowing down its playback to reduce the rebuffering time or speeding up to reduce the end-to-end delay, and 3) adjusting the value of latency limit to decide when to skip the remaining frames in current video segment to reduce the latency. How to make these decisions under stochastic network conditions and with unknown future video information is a fundamental challenge for live streaming applications.

1.1.2 360-degree Video Streaming

Recent advances in the computing resources of machines have prompted interactive applications (e.g., virtual reality) to be viable service types to users. The interaction of these types of applications makes use of head mounted displays (HMD) device to enable 360 video players to act immersively. To approach high-level quality of experience (QoE) for users, it is envisaged that the provisions of high-resolution and 360-degree (or 360 for brevity) videos are essential features. By the fact that conventional video providers (e.g., YouTube) deliver entire video contents to users could generate video traffic up to 40 Mbps [7] , how 360 videos can be conveyed to users with high QoE under constrained network bandwidth determines the futurity of interactive applications.

For fear that the delivery of huge video traffic over the Internet leads to network congestion and QoE degradation, adaptive streaming that adjusts video qualities according

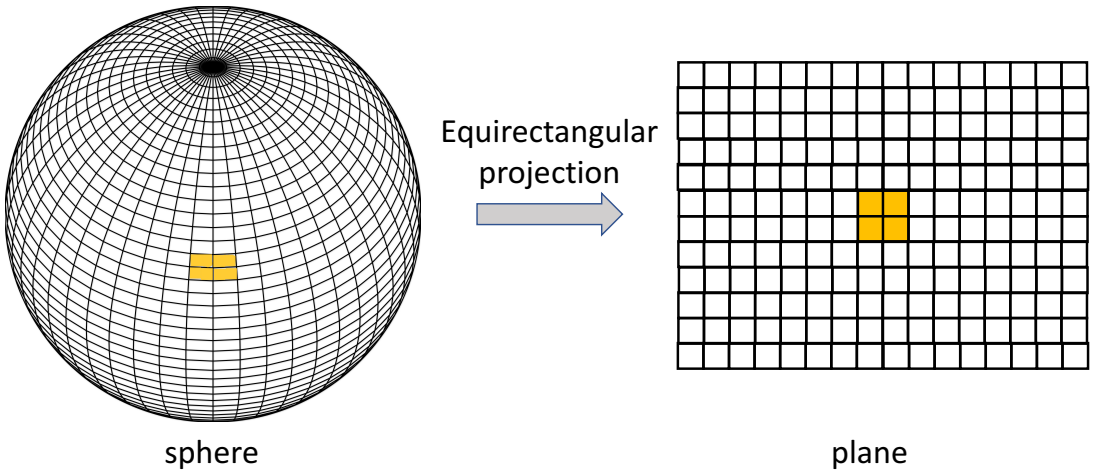


Figure 1.2: The mapping of a spherical surface into a plane.

to users' viewports (i.e., fields of view) is capable of adapting to time-varying network bandwidth. In viewport adaptive streaming, high-quality streams are delivered to viewport areas while reduced qualities are provided to non-viewport ones, since each user can only have relatively narrow fields of view of a 360 video. With the knowledge of users' viewports, the streaming qualities can then be differentiated, thereby saving video traffic from downplaying non-viewport areas [8–10].

Using tiles in viewport adaptive streaming for 360 videos offers a high level of granularity over the projected[1] plane and thus attracts much research attention in the past few years. Each tile is a spatial fragment of a video segment, and it is typically represented by its horizontal and vertical degrees (as depicted in Figure 1.2). In addition, tiles can be encoded by various quality levels according to their respective positions. For instance, we can encode tiles of viewports with high bitrates and reduce video traffic for non-viewport areas. Therefore, tile-based viewport adaptive streaming admits elastic control of the quality levels of tiles for 360 videos, thereby enhancing QoE for users.

To facilitate tile-based viewport adaptive streaming for 360 videos with high QoE,

[1]Each 360 video surrounds a spherical surface, which can be projected onto a two-dimensional plane via a projection method [11] (e.g., equirectangular, cubemap, pyramid projections). In the projected plane, existing video coding that were designed for planar areas becomes applicable, making the design of viewport adaptive streaming for 360 videos much easier than handling spherical surfaces directly.

there are two key issues to be addressed: viewport prediction (VPP) and tile bitrate selection (TBS). The goal of VPP is to predict viewports for the upcoming segments based on head movements, since the interactive applications may guide users to change their facing directions rapidly. Without high-accuracy VPP, severe QoE drops may take place. On the other hand, the mission of TBS is to determine tile bitrates, since the availability of network bandwidth is hightexty dynamic. Large variation among the quality levels of tiles within a viewport can also affect QoE adversely, and hence the smoothness of inter-tile quality levels is another concern. Clearly, how to tackle the VPP and TBS issues plays a vital role in the success of tile-based viewport adaptive streaming for 360 videos.

1.2 QoE Metrics for Video Streaming

Investigating the influencing factors for QoE should be the first step to build high quality video streaming services. Commonly, these influencing factors can be classified into four categories [12]:

- **System level** includes the factors that function in the technical perspective. These factors are related to the characteristics of network (delay, packet loss), terminal devices (operating system, hardware, screen size), and the video streaming applications (ABR scheme design, transmission protocol).
- **User level** factors consider the psychological or physical state of human. These factors consist of gender, age, the interest of users, users' browsing history (whether watched the same or similar video before), and hour of the day.
- **Content level** factors are mainly the basic properties of a video file/sequence such as encoding format, framerate, resolution, encoding rate, duration, type of video, content quality, language used in the video, etc.
- **Context level** considers the impact of the environment of users' QoE. This kind of factors could be the location of the user (flying plane, running train, quite room), or the purpose of watching videos (e.g., education, interview, chatting, entertainment), etc.

This thesis focus on proposing solutions to design better ABR algorithms since ABR strategy (system level factor) is one of the most important technically controllable influencing factors for QoE of video streaming services. Moreover, better ABR algorithms lead to positive influences on several controllable content level factors (e.g., resolution, encoding rate). In the following parts, we introduce several common QoE metrics to measure the performance of ABR algorithms from the subjective and objective perspective.

1.2.1 Subjective QoE Metric.

The most direct way to assess QoE during a video streaming session is to ask a limited number of human subjects to watch videos in a controlled environment and give opinion scores based on their experience. The opinion score is defined as "value on a predefined scale that a subject assigns to his opinion of the performance of a system" by The International Telecommunication Union (ITU) [13]. The mean opinion score (MOS) is the average of these opinion scores which are commonly rated on a 5-point scale: $[1 : bad, 2 : poor, 3 : fair, 4 : good, 5 : excellent]$ [14]. A better ABR algorithm should be able to achieve higher MOS ratings across different users.

MOS has been the most popular subjective QoE metric for video quality assessment. However, MOS might be user biased, since the rating of a user might be influenced by many personal factors such as gender, age, psychological status, interest on video content, etc. In this case, utilizing MOS as the metric is difficult to fairly evaluate different ABR algorithms. Moreover, the measurement of MOS is slow (waiting for subjects to finish watching video sessions) and unscalable (impractical to test on all kinds of network conditions or video sequences). In addition, MOS cannot provide the exact criteria to guide the design of a better ABR scheme.

1.2.2 Objective QoE Metric.

A promising way to quantify QoE is to use objective metrics which can not only reflect users' subjective perceptions and satisfactions but also be unbiasedly and easily obtained [15, 16]. In practice, most existing ABR algorithms for video streaming scenarios

have been designed and evaluated based on several common QoE metrics, that are common across different video applications and have been proven to be critical for the measurement of user perceived quality as well as user engagement: [17–19].

- Video quality: The average per-segment quality over the whole video streaming session. Many existing works have represented video quality with the values of bitrate [20–22], which is the number of bits per second and generally determines the size and quality of a video. Another classic metric is Peak Signal-to-noise Ratio (PSNR), which is calcualted as the ratio between the maximum possible power of a signal and the power of corrupting noise that affects the fidelity of its representation. There are also other metrics to be used to assess video quality, such as Peak Signal-to-Perceptible-Noise Ratio (PSPNR) [2] [23], Structural Similarity Index Measure (SSIM) [3] [24], and Video Multimethod Assessment Fusion (VMAF) [4] [25]. The choice among these metrics usually depends on their calculation simplicity, application scenario, and video types, etc.

- Temporal quality smoothness: Also know as quality fluctuations. This tracks the magnitude of the quality changes from one segment to the next one. Smoother quality changes gives viewers better experience.

- Rebuffering ratio: For each segment rebuffering occurs when the transmission time is longer than the playback buffer occupancy level. Frequent rebuffering events may make users easily abandon watching the current video stream thus leading to a revenue decrease for the content providers.

Different video streaming scenarios aim to provide different kinds of services, thus requiring different sets of QoE metrics and inconsistent preferences on optimizing different QoE metrics. We'll introduce some common QoE metrics for two typical video streaming scenarios (live and 360 video streaming) in the following.

[2] PSPNR is designed to improve PSNR by filtering out the quality distortions that are not perceived by viewers.

[3] SSIM estimates video's quality by measuring the structural similarity between the received video sequences and the original undistorted ones.

[4] VMAF predicts subjective quality based on the received video sequences and the original undistorted ones with machine learning models.

QoE Metrics for Live Streaming Besides the three QoE metrics (video quality, temporal bitrate smoothness, rebuffering ratio) already designed in the VoD services, new QoE metrics need to be considered for live streaming services due to their strict requirement for low latency. In the live streaming grand challenge [6], another two QoE metrics are defined: the latency penalty and the frame skipping penalty. The latency penalty quantifies the influence of the latency on the user perceived quality. Although skipping several frames can reduce the latency, but skipping too many frames may result in discontinuous playback of the video content or losing important information. In this hence, the skipping frame penalty is used as another QoE metric to balance the latency and content completeness. As a summary, an ABR algorithm designed for live streaming scenarios needs to optimize a QoE function consisting five QoE elements.

QoE Metrics for 360-degree Video Streaming 360-degree video streaming applications usually consider four QoE metrics: video quality, temporal quality smoothness, rebuffering ratio, as well as spatial quality smoothness. Since the users can only watch the video content in the viewport area of the whole sphere, the definitions of video quality and temporal quality smoothness will be modified to represent the average per-segment quality of content in the viewport area across all video segments, and the quality variation of the content of the viewport areas between adjacent video segments. A new QoE element, spatial quality smoothness, is proposed for tile-based 360-degree video streaming. The new term represents the quality smoothness of tiles in the viewport area in per-segment, higher quality differences among adjacent tiles will make users see edges of each tile thus leading to bad QoE.

1.3 Challenges

To design systems that optimize QoE in various video streaming scenarios, one should address the challenges exposed by each scenario. In the following, we first introduce two universal challenges that are faced in all video streaming scenarios, then identify the new challenges exposed by live and 360-degree video streaming applications respectively.

1.3.1 Universal Challenges for Video Streaming

The policies employed by ABR algorithms heavily influence video streaming performance. However, these algorithms face two primary practical challenges:

- **Network dynamics.** Network conditions may vary significantly over time while streaming a video. For example, on cellular links where sudden fluctuation may often happen, it's impossible to predict future bandwidth accurately, thus leading to underutilized networks (lower video quality) or large transmission delays (rebuffering). To address this, ABR algorithms should be robust to the inaccurate future bandwidth prediction and make bitrate decisions in the long term run.
- **QoE conflicts.** ABR policies need to handle the trade-off among several QoE metrics such as optimizing video quality (i.e., highest possible bitrate), minimizing rebuffering rates (i.e., scenarios where the client's buffer is empty and playback stops), and reducing video quality fluctuations (i.e., avoiding bitrate changes between adjacent video segments). However, many of these goals may intrinsically conflict with each other. For example, on networks with limited bandwidth, keeping rebuffering at a low rate leads to consistently requesting segments encoded at lower bitrate which will sacrifice quality; on the other hand, maximizing the quality by downloading highest possible bitrate will inevitably result in more rebuffering events. Moreover, on varying networks, greedily selecting the highest bitrate based on estimated available bandwidth may lead to substantial quality fluctuations.

1.3.2 New Challenges for Live Video Streaming

Besides dynamic network environment and conflicting QoE goals, live video streaming applications exposes two new technical challenges:

- **Hybrid actions.** To satisfy the strict latency requirement, a live video player have more complicated actions to make than on-demand video player. For example, as introduced in [6], a live video players needs to make three decisions: 1) bitrate, 2) target buffer level, and 3) latency limit. The action space is larger and actions may have influences among each other.

- **Unknown future video information.** Unlike on-demand video streaming services where videos where videos are pre-recorded and videos' information (e.g. sizes of all video segments) can be pre-fetched from the server, videos are generated on the run in live video streaming scenarios. Video segment sizes are changing along time and can vary largely across different video contents. It's challenging to estimate future video segment sizes for each bitrate level, and inaccurate video segment size predictions can lead to bad video quality or large latency.

1.3.3 New Challenges for 360-degree Video Streaming

To reduce bandwidth consumption, tile-based 360-degree video streaming method is proposed, which exposes two new challenges:

- **Unknown future viewport.** Different users have different head rotation preferences on different video contents. Many factors can influence which part of the video users will watch in the future. For example, users may prefer to explore the video at the beginning then feel tired and keep stable; users may move his head to follow a player or the ball while watching a football game. It's difficult to predict accurately where a user will rotate his head in the future, and inaccurate viewport prediction will decrease the QoE of the users.

- **Large action space.** For regular video, the ABR policy only needs to choose an approriate bitrate level for one video segment. However, for tile-based 360-degree video, tile bitrate decisions need to be made for each tile in a video segment, different tiles may have different bitrate levels. For a streaming scenario where there are N tiles in one video segment and M available bitrate levels for each tile, the action space will be M^N.

1.4 Learning as a Solution for Video Streaming

Inspired by the reality that machine learning techniques have been successfully applied in many research fields (e.g. robotics, computer games, VoD video streaming), this

dissertation develops learning-based solutions to tackle the challenges in various video streaming scenarios. The core advantages of machine learning techniques are that they can learn latent patterns underlying the large amount of data traces (network throughput, head motion, video), and make accurate prediction or good decisions based on the learned data patterns. Next, we briefly introduce the three solutions proposed in this dissertation.

- **Learning-based Live Video Streaming** (Chapter 2). Existing learning- [20] or heuristic-based work [21, 22] on VoD video streaming scenarios cannot take hybrid actions and lacks the component for future video information prediction. Prior work on live video streaming [26–28] are based on human heuristics and unable to appropriately handle the dynamics of network and video and trade-off among different QoE metrics. To address the challenges with the power of machine learning, we propose HD3 (Distributed Dueling DQN with discrete-continuous Hybrid action spaces), a novel deep reinforcement learning (DRL) algorithm which makes both discrete (bitrate and target buffer level) and continuous (latency limit value) actions. In detail, we use HD3 to train a neural network agent to make decisions based on environment states (e.g. past network throughputs, current buffer level, historical video frames' sizes) to optimize users' QoE (i.e., high video quality and low end-to-end delay). We did extensive simulations on real-world video datasets and various network conditions, the reuslts show that the proposed scheme HD3 can outperform other state-of-the-art DRL schemes in different network conditions and video scenes.

- **Learning-based 360-degree Video Streaming** (Chapter 3). Either learning or heuristic-based studies on VoD video streaming services are designed for a small action space and lacks a well-designed QoE function, which make them not applicable in tile-based 360-degree video streaming. Existing studies [29–32] on 360-degree video streaming are designed based on heuristics which cannot perform well under dynamic environments and multiple conflicting QoE goals. To leverage of power of learning to address the challenges, we first divide the tiles into several classes to reduce the action space, then utilize a state-of-the-art DRL algorithm-A3C [33] to train a neural TBS agent to map the environment states (e.g.

past network throughputs, current buffer level, future video tile sizes) to the bitrate for each tile class. We did extensive simulation on real-world network, video, and head motion datasets, and the results demonstrate that the proposed scheme can achieve better QoE than the comparing schemes.

- **Sinusoidal Viewport Prediction for 360-degree Video Streaming** (Chapter 4). To improve viewport prediction accuracy, existing studies are in a trend of utilizing more information (e.g. video content [34, 35], or other users' motions traces [30]) or designing more complicated models ([36]). However, these approaches will potentially give rise to excessive computational overhead and latency. To achieve both high accuracy and low overhead, we propose a sinusoidal viewport prediction (SVP) system which leverages sinusoidal values of rotation angles to predict orientation, and utilizes the relationship between prediction errors, prediction time window and head movement velocities to further improve the system performance. Based on the simulations conducted on multiple real-world bandwidth, head motion, and video datasets, we show that the proposed scheme outperforms state-of-the-art comparison schemes under various buffer thresholds and bandwidth settings in terms of viewport prediction accuracy and video quality.

The key contribution of this dissertation is a suite of learning-based systems to optimize users' QoE in both live and 360-degree video streaming scenarios (a quick overview can be found in Figure 1.3). In particular, we identify key challenges to QoE optimization, tackle these challenges by novel system and algorithm designs that leverage deep reinforcement learning for time-series decision and other machine learning techniques for prediction, and perform extensive simulations on real-world datasets to demonstrate that our proposed solutions can substantially improve the QoE in both live and 360-degree video streaming scenarios.

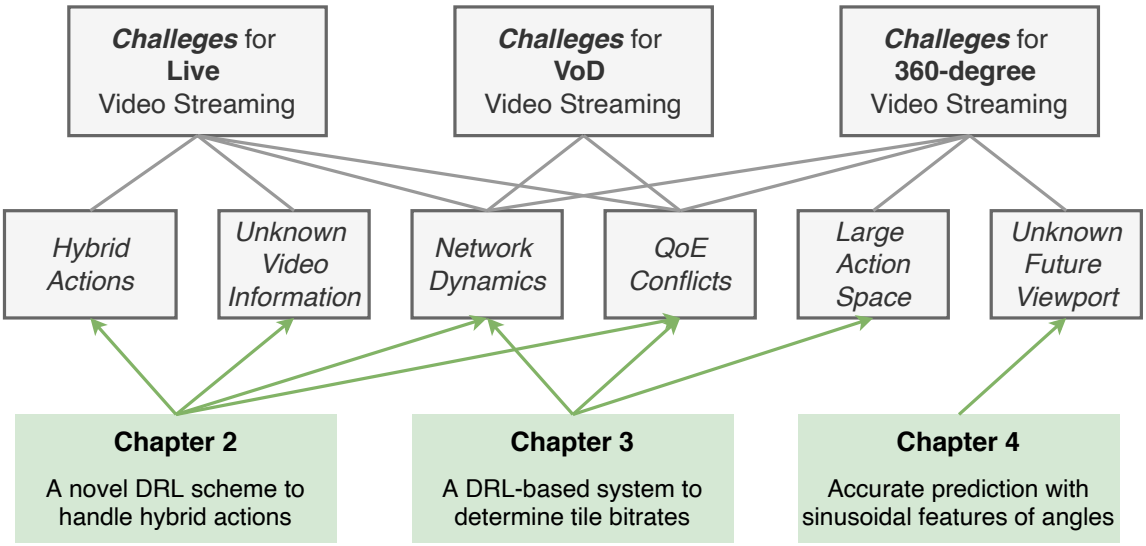


Figure 1.3: The main contribution of this dissertation is to develop three learning-based solutions (bottom) to address the key challenges (middle) exposed by live and 360-degree video streaming scenarios (top).

1.5 Organization

The rest of this dissertation is organized as follows. In Chapter 2, we introduce the of existing ABR schemes for live streaming and 360-degree video streaming, as well as related work for viewport prediction systems. Chapter 3, 4, and 5 describe three important components of this dissertation: (1) The learning-based system optimizes user QoE in live video streaming scenario by designing distributed Dueling DQN algorithm to solve hybrid discrete and continuous action space problem; (2) A learning-based system optimizes users' QoE in 360-degree video streaming scenario by applying DRL algorithm to train a neural agent to select bitrates for each tile; (3) The sinusoidal viewport prediction system predicts future viewport by utilizing sinusoidal values of head rotations.

Chapter 6 summarizes the contributions of the dissertation, discusses the lessons learned and limitations of the proposed solutions, and ends with several future perspectives.

2 Related Work

2.1 Live Video Streaming

In this part, we first introduce the existing ABR algorithms for VoD streaming, then introduce the related work in live streaming. Although the ABR algorithms designed for VoD streaming cannot be directly applied for live streaming, the idea behind these schemes have inspired the design of ABR and latency control algorithms in live streaming.

2.1.1 ABR algorithms for VoD services

Existing ABR algorithms for VoD applications can be primarily categorized into three classes: rate-based, buffer-based, and quality-based. The rate-based approach FESTIVE [37] first predicts future bandwidth by the harmonic mean of the past measured throughputs, then select the highest possible bitrate the bandwidth can support.

Conversely, buffer-based methods [22, 38] make bitrate decisions for future segments solely based on the client's playback buffer level. The goal of these schemes is to keep the buffer occupancy at a target level which balances video quality and rebuffering. BBA [38] maps client's buffer level to bitrate through a pre-configured piece-wise linear function. BOLA [22] formulates the ABR problem as an optimization problem and solve it with the Lyapunov function.

Quality-based schemes [20, 21] try to optimize users QoE by making bitrate decisions based on both estimated future bandwidth and current buffer level. MPC [21] applies model predictive control approaches which utilize both bandwidth estimations and buffer level information to make appropriate bitrate decisions to maximize QoE over a horizon of several future segments. Pensieve [20] leverages DRL to train an neural ABR agent to optimize QoE using multiple input useful signals including past measured bandwidths and buffer level.

All algorithms above are designed for VoD services which cannot be directly be applied to tackle the new challenges exposed by live video streaming scenarios. More specifically, these algorithms do not encode latency into their QoE optimization function thus leading to high latency in live streaming services. Moreover, they are not designed to take only bitrate actions and it's difficult to extend them to take more actions (e.g. bitrate level, target buffer level, latency limit).

2.1.2 ABR algorithms for Live streaming

Many existing works [26–28, 39–41] for live streaming have utilized techniques of playback control or frame dropping to reduce latency. Earlier schemes are mainly designed based on human heuristics. Li *et al.* [26] utilize the Group of Pictures based cumulative average jitter to determine the playback threshold. In [27, 28] the technique of frame dropping is utilized by heuristic-based methods. However, there is still space to reduce latency beyond these schemes.

Recently, Yi *et al.* [39] held a live streaming grand challenge which encourages researchers to apply machine learning algorithms to design efficient ABR algorithms and latency control schemes based on both the technique of playback control (by making

decisions among several discrete target buffer levels) and frame skipping (controlled by setting a continuous latency limit value). To tackle the problem proposed in this challenge, several solutions has been proposed and published. Hong *et al.* [41] applied Deep Deterministic Policy Gradient, a popular DRL algorithm on continuous tasks, to train an ABR agent which can output only continuous values, then map some of these actions to discrete values for making bitrate decision and determining target buffer level for playback control. Peng *et al.* [40] proposed a hybrid control scheme based on heuristic playback rate control, latency-constrained bitrate adaptation and QoE-oriented frame dropping. This scheme will first discretize the continuous latency limit values into multiple discrete values, then choose the best one from the discretized values.

The schemes introduced above is summarized in Table 2.1. The learning-based schemes can usually outperform heuristic-based schemes by better adapting to the network dynamics. On the other hand, learning-based algorithms often lead to higher overhead. The two recent work [40, 41] cannot output hybrid (discrete and continuous) actions. To tackle the same challenge, we have proposed a novel DRL-based scheme HD3 (see Chapter 3) [42] which are designed based on the Dueling DQN algorithm and can output hybrid actions.

Table 2.1: Summary of Related Work on Live Video Streaming

Work	Algorithm	Latency Control	Action	Adaptivity to Dynamics	Overhead
		VoD video streaming			
[37]	Heuristic	-	Discrete	Low	Low
[38]	Heuristic	-	Discrete	Low	Low
[21]	MPC	-	Discrete	Medium	Medium
[22]	Optimization	-	Discrete	Medium	Medium
[20]	DRL (A3C)	-	Discrete	High	High
		live video streaming			
[27]	Heuristic	frame skipping	Discrete	Low	Low
[26]	Heuristic	playback control	Discrete	Low	Low
[28]	Heuristic	frame skipping	Discrete	Low	Low
[40]	MPC	playback control & frame skipping	Discrete	Medium	Medium
[41]	DRL (DDPG)	playback control & frame skipping	Continuous	High	High

2.2 360-degree Video Streaming

Tile-based viewport adaptive 360 video streaming has been an emerging method in the research society. Many researchers have been proposing their schemes. Graf propose three tile based schemes to compare their performance [43]. Their schemes are called full delivery basic, full delivery advanced and partial delivery. For full delivery basic, they assign the highest possible bitrate for the tiles in the viewport area, and the lowest bitrate for the other tiles. For full delivery advanced, they still try to give the viewport tiles a highest possible bitrate, give the other tiles a lower but not lowest bitrate. For partial delivery, they only download the viewport tiles at a highest bitrate, but leave all the other tiles blank. Their approaches are based on very trival heuristic. [44] proposes a scheme which split the tiles into three areas: viewport, adjacent, and outside areas. Then try to assign the highest bitrate to the tiles at the order of viewport, adjacent, outside areas within the available bandwidth budget. However, all their schemes are based on fixed heuristics, can not get good performance under the varying network conditions. However, these methods only considers video quality but ignores the other QoE goals such as spatial and temporal tile bitrate smoothness, and rebufferring.

Recently, more advanced algorithms [29–32, 45–47] are proposed. He *et al.* [45] design a MPC-based optimization framework which optimizes the QOE of multiple segments in a future time window. Zhang *et al.* [47] apply DRL to train a LSTM based neural agent to determine the bitrate for only the viewport tiles. Zhang *et al.* [46] utilize Beam search optimization to allocate rates for tiles to optimize QoE.

Other researches [29–32] design tile-based steaming schemes in a two-step form. These algorithms first determine a total bitrate budget for the next segment based on available information (e.g., bandwidth estimation, current buffer level), then allocate bitrates for each tile in the next segment to optimize QoE. In [29–31], authors utilize simple heuristics to calcualte the total bitrate budget for next segment, while in [32] Guan *et al.* use MPC based method to get the bitrate budget. Then to allocate rates for tiles, [30] proposes a Multiple-Choice Knapsack based solution, [31] performs exhaustive search after classifying tiles into several classes, [32] design a simple heuristic based method.

All the schemes introduced above is summarized in Table 2.2. These schemes can

Table 2.2: Summary of Related Work on 360-degree Video Streaming

Work	Algorithm	Adaptivity to Dynamics	Prediction Error Handling	Quality Smoothness	Overhead
[43]	Heuristic	Low	No	-	Low
[44]	Heuristic	Low	Yes	-	Low
[29]	Optimization	Medium	Yes	Spatial	Medium
[30]	Heuristic	Low	Yes	-	Low
[45]	MPC	Medium	No	Spatial & Temporal	Medium
[31]	MPC	Medium	Yes	Spatial & Temporal	Medium
[47]	DRL (A3C+LSTM)	High	No	Spatial & Temporal	High
[32]	MPC	Medium	Yes	Spatial & Temporal	Medium
[46]	Beam Search	Medium	Yes	Spatial & Temporal	Medium

be compared between each other in terms of their adaptivity to the dynamic network conditions, whether they have explicitly handled the possible viewport prediction error, whether their scheme can achieve spatial or temporal quality smoothness, as well as the overhead of the models. We have proposed a DRL based ABR scheme (see Chapter 4) for 360-degree video streaming, the core difference between our shceme and a recently proposed DRL-based algorithm ([47]) is that our proposed system can handle possible viewport prediction error by dividing tiles into three areas and selecting bitrates for each area.

2.3 Viewport Prediction

Viewport prediction is an important technique that can be utilized by various applications, e.g. augmented reality (AR), virtual reality (VR) and volumetric video streaming [48, 49]. Viewport prediction can be categorized into two classes: trajectory- and content-based methods. Trajectory-based methods predict future viewport either with user's own (single-user) or other users' (cross-user) historical head rotations. Single-user methods usually estimate the user's future head rotations by solving a regression problem. Qian

et al. [50] use logistic regression to predict future viewport. Lan *et al.* [29] propose a probabilistic model assuming the prediction error follows a Gaussian distribution. Jiang *et al.* [51] apply a long short term memory (LSTM) based model to predict future head rotations. Zhang *et al.* [36] utilize an ensemble of three LSTM models to further improve the prediction accuracy. However, all these methods are not accurate in the *yaw* direction due to the periodicity issue.

Cross-user methods assume that users have similar region-of-interest (ROI) when watching the same video, and hence that it is possible to exploit multi-users' ROI behavior to predict viewport. Lan *et al.* [30] group users with density-based spatial clustering of applications with noise (DBSCAN) in the server, then on the client end, classify the user to the corresponding cluster with a support vector machine (SVM) classifier, and finally obtain the viewing probability from the cluster. Ban *et al.* [52] first predict future fixations with LR, then utilize K-Nearest-Neighbor (KNN) to find the K nearest fixations of other users around the LR result to improve accuracy. Petrangeli *et al.* [53] first identify user clusters with a kind of spectral clustering algorithm, then fit a regression model for each cluster, finally predict with the regression model from the user's corresponding cluster. Nasrabadi *et al.* [54] first cluster users based on their quaternion rotations, then classify the target user to the corresponding cluster and estimate the future fixation as the cluster center. If no available cluster for the target user, the last sample will be used as the future viewport. Although these methods can have relatively high accuracy in the long term, it cannot be deployed in a live streaming scenario where no other users have watched the same video before. Moreover, cross-user methods can perform worse than single-user methods on short term prediction.

To improve the prediction accuracy, content-based methods utilize both rotations and video contents as features. Zhang *et al.* [35] proposed a generative adversarial network (GAN) to generate multiple future frames conditioned on the single current frame and then anticipates corresponding future gazes in the upcoming few seconds. Xu *et al.* [55] proposed a deep reinforcement learning (DRL) based approach to better model the users' attention with video contents. Many works [34, 56–58] design a hybrid architecture of CNN and LSTM models. They use a convolutional neural network (CNN) to extract video content features from the saliency maps or the original images, and use LSTM to extract

motion patterns from history rotations. Then, they predict the future viewport based on CNN and LSTM features. Recently, some works are conducted to design content-based methods for live virtual reality (VR) video streaming. Feng *et al.* [59] utilize optical flow and Gaussian mixture model (GMM) for motion detection and feature tracking, then predict user's future viewport by leveraging a dynamic user interest model. Feng *et al.* [60] leverage CNN based model to predict future viewport in live streaming by modifying the training/testing process and the workflow of the CNN application. To further improve the prediction accuracy, Feng *et al.* [61] employ a hybrid deep learning model which involves both CNN and LSTM models. All these methods would have a burden to the practical deployment in a real system since they consume excessive computing resources.

We summarize the existing schemes in Table 2.3. High overhead or cross users based prediction make the models difficult to be deployed in live streaming scenarios (where quick prediction is needed) or at mobile devices (where computational resources are limited). To overcome the above issues, we proposed a sinusoidal prediction system (see Chapter 5) which has low overhead and high prediction accuracy.

Table 2.3: Summary of Related Work on Viewport Prediction

Work	Algorithm	Available on Mobile Devices	Available for Live Streaming	Overhead	Accuracy
Trajectory-based Single-user (with periodicity issue)					
[50]	LR	Yes	Yes	Low	Low
[29]	LR	Yes	Yes	Low	Low
[51]	LSTM	Yes	Yes	Medium	Low
[36]	LSTM Ensemble	Yes	Yes	Medium	Low
Trajectory-based Cross-user (with diverse interest and periodicity issue)					
[30]	Clustering	No	No	Low	Low
[52]	Clustering	No	No	Low	Low
[53]	Clustering	No	No	Low	Low
[54]	Clustering	No	No	Low	Low
Content-based (with exceeding computational overhead issue)					
[55]	DRL	No	No	High	High
[34]	CNN & LSTM	No	No	High	High
[35]	GAN	No	No	High	High
[56]	CNN & LSTM	No	No	High	High
[57]	CNN & LSTM	No	No	High	High
[58]	CNN & LSTM	No	No	High	High
[59]	GMM	No	Yes	High	High
[60]	CNN	No	Yes	High	High
[61]	CNN & LSTM	No	Yes	High	High

3 Learning-based Live Video Streaming

3.1 Overview

Live streaming applications are becoming increasingly popular recently, and it exposes new technical challenges compared to regular video streaming. High video quality and low latency are two main requirements in live streaming scenarios. A live streaming application needs to make bitrate and target buffer level decisions as well as sets a continuous latency limit value to skip video frames.

In this chapter, we propose HD3 (Distributed Dueling DQN with discrete-continuous Hybrid action spaces), a novel deep reinforcement learning (DRL) algorithm which makes both discrete (bitrate and target buffer level) and continuous (latency limit value) actions, to achieve high video quality and low end-to-end delay. In detail, we use HD3 to train a neural network agent to make decisions based on environment states (e.g. past network throughputs, current buffer size, historical video frames' sizes) to optimize users'

QoE. The agent is trained and evaluated with various network environments and video scenes. The simulation results demonstrate that our scheme can generate a single agent which can obtain good performance under all different network conditions. In addition, HD3 can converge in a relatively short time with multiple processes, and that will be extremely helpful for the industry to save training time where millions of data traces exist. The main contributions of this work can be summarized as follows:

- We design a novel DRL algorithm-HD3 which can solve problems with discrete-continuous hybrid action spaces.
- Our scheme generates a single agent which can perform well under various network conditions.
- The distributed scheme can converge fast and be applied at scale.

3.2 System Design

3.2.1 Reinforcement Learning for Live Streaming

In this part, we first give an overview of reinforcement learning framework, then introduce how we formulate the live streaming problem into the reinforcement learning framework.

Our scheme utilizes reinforcement learning to generate a neural network agent from the observations instead of designing fixed rules based on heuristics. Figure 3.1 shows how the agent will interact with a live video player. At each time step t, the agent takes an action a_t based on the state (e.g., past network throughputs, current buffer size, historical video frames' sizes) observed from the environment (live video player). Then the environment transits to the next state s_{t+1}, calculates the reward based on the QoE metrics and sends a reward signal r_t to the agent. The goal of RL is to maximize the sum of discounted reward $\mathbb{E}\left[\sum_{t=0}^{n} \gamma^t r_t\right]$, where $\gamma \in (0, 1]$ is a discounting factor. The RL algorithms train and update the parameters of the neural network agent to achieve this goal. Next, we explain how we design the state, action space and reward function in detail.

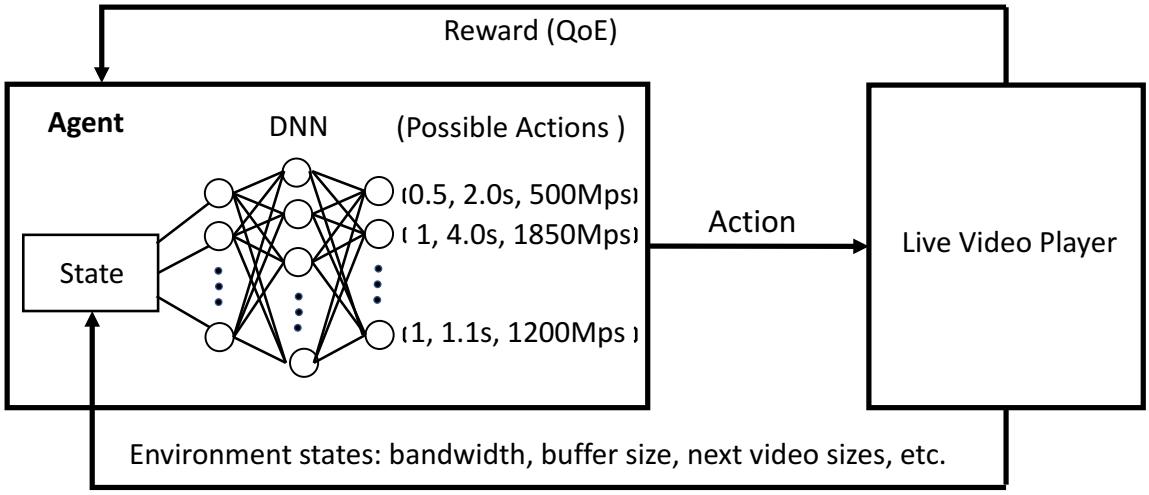


Figure 3.1: The Workflows of RL.

State: After downloading each GOP (group of pictures) segment, the agent observes the state s_t (see Figure 3.2) from the environment and takes it as input to the neural network, where $s_t = (\vec{x_t}, \vec{\tau_t}, \vec{cf_t}, \vec{sk_t}, \vec{s_m}, b_t, r_t, bf_t, c_t)$. In time step t, we use x_t, τ_t, sk_t to represent the total throughput, delay, skip time in the last GOP segment. For the past k GOP segments, the historical information of throughput, delay and skip time are denoted by $\vec{x_t}, \vec{\tau_t}$ and $\vec{sk_t}$, respectively. $\vec{cf_t}$ is the cdn buffer flags in the past k frames which may reflect the network status of remote server or video broadcaster. b_t denotes current buffer size and r_t represents the bitrate in last downloaded GOP segment. bf_t is local buffer flag indicating whether the video player is rebuffering, and c_t is buffer size in the CDN server.

For the next GOP segment, $s_1, s_2, ..., s_m$ denotes the GOP size for each available bitrate respectively. However, the future video size might not be directly observed from the CDN buffer, then we need to estimate. After downloading each GOP segment of bitrate r with size s_r, we first estimate the next GOP segment's size of bitrate r as s_r. Then we calculate the GOP segments' sizes of other bitrate based on fixed ratios between different bitrates. By analyzing the video dataset (on Section 3.3.2), we find that the GOP sizes for any two different bitrates at all time steps have almost fixed proportion with little variance.

Action: In this task, we need to take hybrid actions including both discrete and continuous actions. More specifically, we need to set a continuous scalar value for the

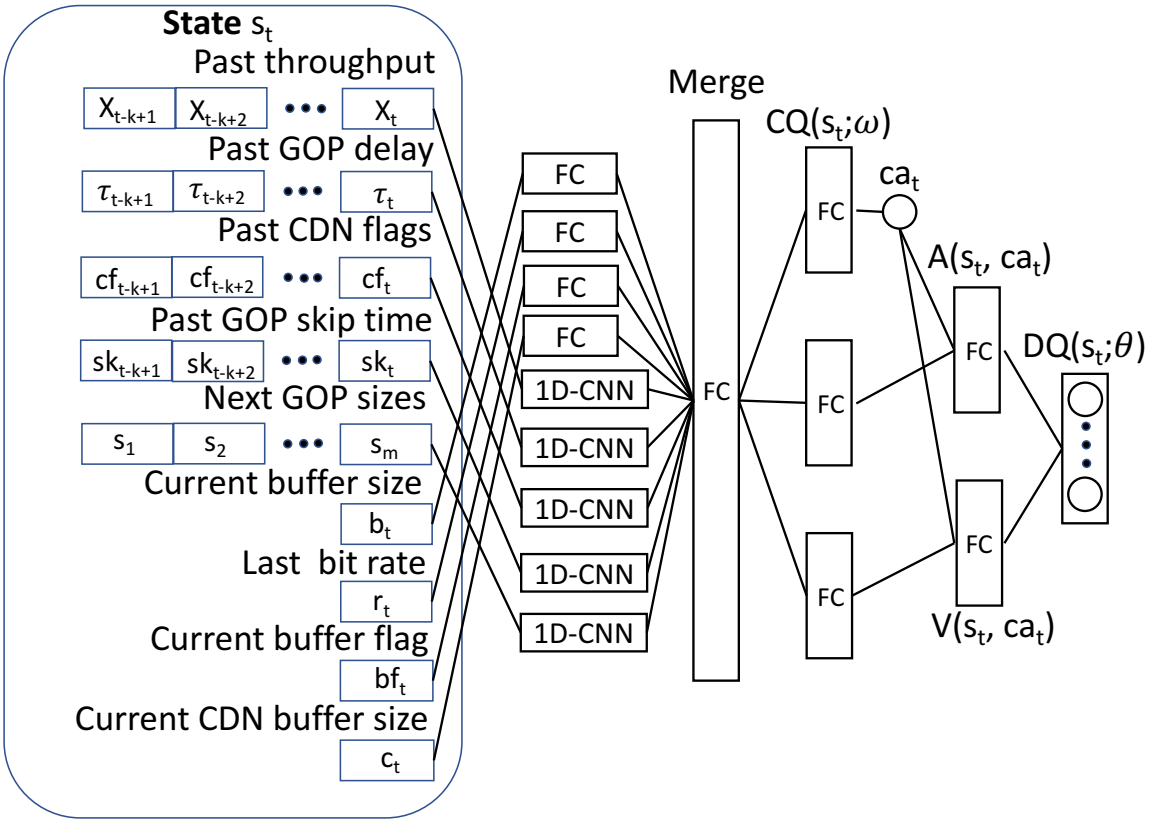


Figure 3.2: HD2 Architecture

latency limit and make choices from two sets of discrete available bitrate levels and target buffer levels. The hybrid action space makes this problem more challenging than other problems where only discrete or continuous actions need to be considered. The settings for the three actions can be set based on the application scenario and user preference. The implementation of these actions in this work is shown on Section 3.3.4.

Reward: The definition of reward (QoE) function needs to take all the QoE metrics (e.g. video quality, latency, bitrate fluctuation) into consideration. The equation of reward function is shown on Section 4.3.5.

Algorithm 3.1 HD2

Initialize function DQ with random weights;
Set the capacity of replay memory $\mathcal{D}$ as N;
for *episode = 1, M* **do**
  Set live video player state as s_1
  for *t = 1, T* **do**
    With probability ϵ select random actions da_t and ca_t ;
    Otherwise select $ca_t = CQ(s_t; \omega)$, and $da_t = \max_{da}(DQ(s_t, da; \theta))$;
    Take action (ca_t, da_t) in simulator, obtain reward r_t and collect video player information s_{t+1} ;
    Store transition $(s_t, ca_t, da_t, r_t, s_{t+1})$ in $\mathcal{D}$;
    Sample random minibatch of transitions $(s_j, ca_j, da_j, r_j, s_{j+1})$ from $\mathcal{D}$;

$$y_j = \begin{cases} r_j & s_{j+1} = terminal \\ r_j + \gamma \max_{da} DQ(s_{j+1}, da; \theta) & otherwise. \end{cases}$$

    Compute gradient based on $(y_j - DQ(s_j, da_j; \theta))^2$ according to equation 2, and update θ;
  end
end

3.2.2 HD2: Dueling DQN with Hybrid Action Space

Now, we will introduce our proposed scheme HD2 in detail. The conventional Dueling DQN [62] scheme can only tackle the problem with discrete action spaces. To cope for the live streaming task which have hybrid action spaces, we design a new network architecture (as shown in Figure 3.2) based on the conventional one. Next we will first explain a typical DQN scheme, then introduce the architecture design of dueling DQN with hybrid action spaces.

After downloading a GOP video segment, the agent will get a reward r_t from the simulator. The goal of a RL algorithm is to select the best action sequences which can maximize the sum of received rewards. DQN [63] utilizes a neural network to construct a discrete action value function $DQ(s, a; \theta)$ to estimate the return (or long term rewards) the agent can receive under any state action pairs. Then in any state s, the agent will select the action a which yields the largest value.

To achieve an optimal discrete action value function $DQ(s, a; \theta)$, the agent needs to observe a sequence of state and action pairs, and store them into a replay memory $\mathcal{D}$, then get the loss function based on the experiences (s_t, a_t, r_t, s_{t+1}) randomly sampled from the replay memory $\mathcal{D}$ as follows:

$$L(\theta) = \mathbb{E}_{s,a}[(y - DQ(s, a; \theta))^2] \tag{3.1}$$

where $y = \mathbb{E}_{s'}[r + \gamma max_{a'} DQ(s', a'; \theta')]$ is the target for $DQ(s, a; \theta)$. Then we can compute the gradient by differentiating the loss function above:

$$\nabla_\theta L(\theta) = \mathbb{E}_{s,a}[(y - DQ(s, a; \theta))\nabla_\theta DQ(s, a; \theta)] \tag{3.2}$$

HD2 architecture (as shown in Figure 3.2) is designed based on Dueling DQN architecture [62], where the final layer $DQ(s, a; \theta)$ is produced by three sequences of fully connected (FC) layers $CQ(s; \omega)$, $A(s, ca, a; \theta)$, $V(s, ca; \theta)$. First the FC layer $CQ(s; \omega)$ generate the value ca as the continuous action , then the other two FC layers $A(s, ca, a; \theta)$ (denoting the advantage function of state action pairs) and $V(s, ca; \theta)$ (denoting the value function of state) take both the state s and continuous action value ca as input. Finally, we can get the discrete action layer $DQ(s, ca, a; \theta)$ as follows:

$$DQ(s, a; \theta) = V(s, ca; \theta) + (A(s, ca, a; \theta) - \frac{1}{|\mathbb{A}|} \sum_{a\prime} A(s, ca, a\prime; \theta)) \tag{3.3}$$

Since ω is subset of θ, while updating θ the value of ω will also be updated to generate a better continuous action. The pseudocode of HD2 is shown in Algorithm 3.1. The design of HD2 is inspired by [64], however, HD2 poses a single network architecture (instead of two as in [64]) which leads to much smaller computation load and can perform better in live streaming task.

3.2.3 HD3: Distributed HD2

To accelerate the training phase and make our scheme be able to be applied at scale, we propose HD3-a distributed architecture for HD2. The basic idea of HD3 is to collect

Algorithm 3.2 HD3-Actor

```
for episode = 1, M do
    Initialize video player information s_1 ;
    for t = 1, T do
        Follow the same flow as in HD2, get transition (s_t, ca_t, da_t, r_t, s_{t+1}) and save it in
        local buffer LB ;
        if LB.Size() > LB.Capacity then
            Send transitions in LB to Learner ;
            Clear local buffer LB ;
            Obtain latest network parameters from Learner ;
        end
    end
end
```

Algorithm 3.3 HD3-Learner

```
Initialize function DQ with random weights ;
Set the capacity of replay memory D as N ;
while Training do
    Obtain transitions T from each available Actor ;
    for each transition t in T do
        Store t into replay memory D ;
        Sample random minibatch of transitions from D ;
        Calculate loss and update θ with the same method as in HD2 ;
    end
    Send updated network parameters to each Actor ;
end
```

more experience data by distributing the generation of experience traces into multiple Actor agents, and use one Learner agent to compute the gradient of the neural network parameters based on the collected experiences from the Actor agents. The pseudocode of Actor and Learner agents is described in Algorithm 3.2 and Algorithm 3.3 respectively. The distributed algorithm is inspired by [65, 66].

3.3 Evaluation

3.3.1 QoE Definition

In order to evaluate the proposed ABR and latency control algorithms, Gang *et al.* [6] consider five QoE metrics: three commonly used metrics (video quality, rebuffering time, smoothness), and two newly introduced metrics (latency and frame-skipping time) which are particularly for live streaming. Since live streaming has strict requirement on latency, then latency is the most important QoE metric in live streaming scenarios. Using frame-skipping time as a new metric is to handle the trade-off between reducing latency and keeping the continuity of the playback.

Reward Function. The reward function is given as follows:

$$QoE = \sum_{n=1}^{N}\sum_{m=1}^{M}(\beta R_{n,m} - \gamma T_{n,m} - \delta L_{n,m} - \theta S_{n,m}) - \sum_{n=1}^{N-1}\alpha|R_{n+1} - R_n| \tag{3.4}$$

where N is the number of GoPs; M is the number of frames in each GoP. $R_{n,m}$, $T_{n,m}$, $L_{n,m}$ and $S_{n,m}$ mean the bitrate, rebuffering time, latency and frame-skipping time of frame m in GoP n, respectively.

To decide the values of $\beta, \gamma, \delta, \theta, \alpha$, Gang *et al.* [6] run multiple experiments to evaluate several simple ABR algorithms with different combinations of values of these coefficients. Then they choose the combination under which the performance of these ABR algorithms can best reflect their performance in the production system, or in other words, if algorithm A is better than B in the production system, then algorithm A should remain better (or even by similar gap) than B under the evaluation with a good combination of values of the coefficients. Finally, they set the coefficients as follows: (i) β, is set to be the playout time of each frame; (ii) γ , is set to be the max bitrate level (1.850 Mbps); (iii) δ is 0.005 when delay is lower than 1 second, otherwise, δ is set as 0.01; (iv) θ is set to be the lowest bitrate level (0.5 Mbps). Thus, the penalty of frame-skipping is lower than rebuffering; (V) α is set as 0.02.

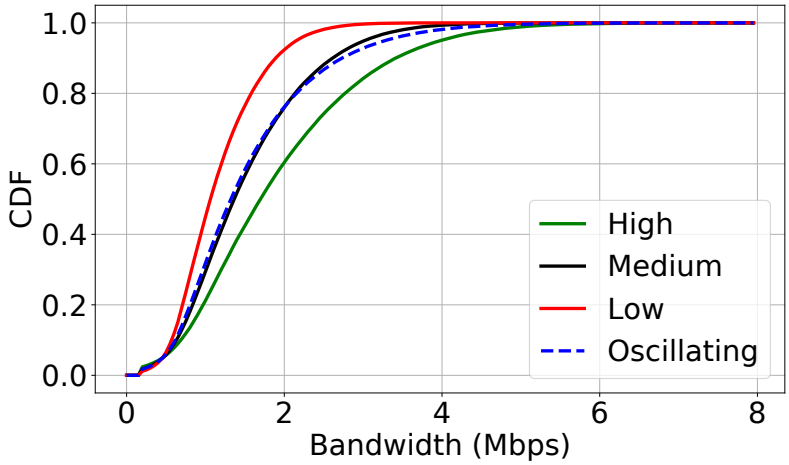


Figure 3.3: CDF of Bandwidth in Four Network Conditions.

3.3.2 Dataset Analysis

Network: The network traces are provided by the grand challenge [6]. There are four kinds of network conditions: high, medium and low throughput, and oscillating conditions (or the mix of high, medium and low conditions). In the dataset, the throughput is recorded every 0.5s. Figure 3.3 shows the CDF of throughput in four types of network conditions. It will be challenging to train a single model which can operate well under so different network environments.

Video: Videos are also given by the competition [6]. Video traces are from three live streaming scenarios: room, game and sports. Each video is encoded into 4 bitrates in the range of [0.5, 0.85, 1.2, 1.85] Mbps. Each video file records the type and size of each frame, and the frame's arriving time at the CDN server.

By analyzing the video datasets in detail, we observe some interesting video's properties which are helpful for predicting future video size in Section 3.2.1. We plot the first 200 GOP sizes of four bitrates from one video in Figure 3.4, where we find that different bitrates of videos have similar changing patterns of GOP sizes along the time. Figure 3.5 more clearly shows the GOP sizes' ratios between adjacent bitrates. Now we can get one insight that ratios between any two bitrates are almost stable with little variance. To further verify this hypothesis, we compute the GOP size's ratios between

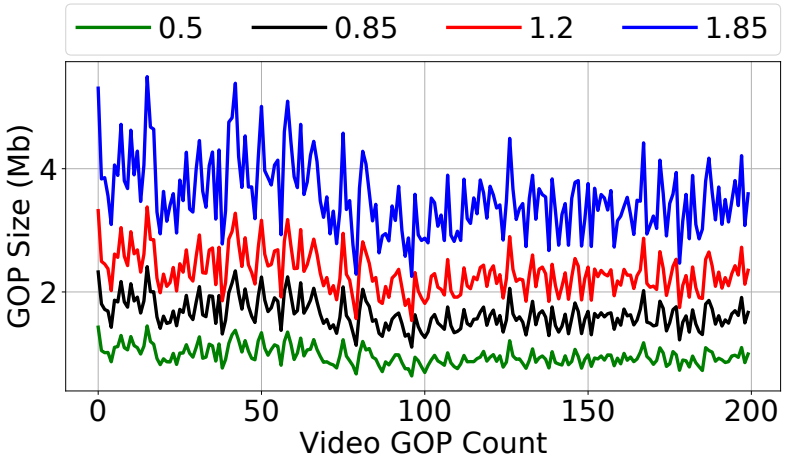


Figure 3.4: GOP sizes of different bitrates from the same video

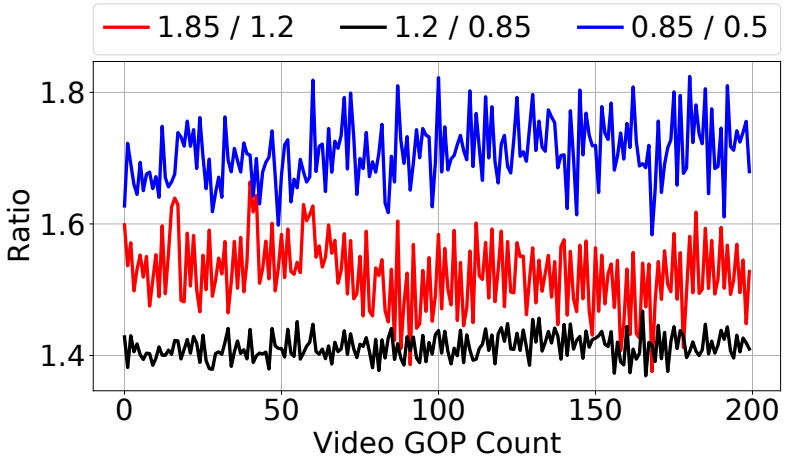


Figure 3.5: Ratios of GOP sizes between adjacent bitrates from the same video

adjacent bitrates from all the videos in all different scenarios, and find that the ratios are all around 1.7, 1.5, 1.4 respectively. Then in Section 3.2.1, we utilize this insight to estimate the GOP sizes of other bitrates based on the size of downloaded bitrate.

3.3.3 Comparison Schemes

Now we introduce several state-of-the-art DRL algorithms which we have used as the baseline schemes in this grand challenge.

- A3C [66]: A general framework consisting of actor and critic networks. A3C uses the technique of policy gradient to update the agent's parameters. It has been applied to train an ABR agent in other video streaming tasks such as in [67–69]. A3C can only handle problems with discrete action spaces.
- DQN [63]: A popular Deep Q Learning algorithm which has achieved good performance on Atari games. It trains a neural network to learn the optimal Q function based on the technique of Bellman equation.
- DDQN [70]: Double DQN, a variant of DQN, has been shown to be able to achieve better performance than DQN in some scenarios. Here we want to try whether DDQN can lead to better performance in a live streaming task.

3.3.4 Implementation

We follow the same settings as [6] for the three actions considered in our work. The latency limit value should be within the range of $[0s, 20s]$, the set of available bitrate levels is $[0.5Mbps, 0.85Mbps, 1.2Mbps, 1.85Mbps]$, and the set of available target buffer levels is set as $[0.5s, 1.0s]$.

The parameter settings and other implementing details are as follows. We set $k = 16$ to collect the past 16 GOPs' information as input. In the neural network, the 1D-CNN has 128 filters, each has size of 4 (except the one taking video sizes as input whose filer size is 3) with stride 1, and all the other full connected layers have hidden size as 128. The learning rate is set as 0.0001, the capacity of replay memory is 10000, and the batch size is 128. We use 12 actors to collect experience on multiple CPU cores, and 1 learner to compute gradients on GPU. It takes about 3 hours to converge with 12 actors, and more than 20 hours with 1 actor. We train the model with all kinds of network and video datasets mixed, since our goal is to generate one single model that can perform well on all scenarios.

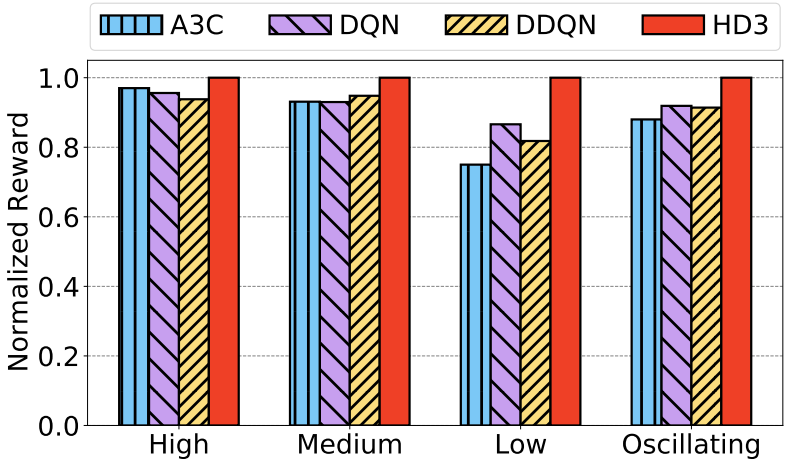


Figure 3.6: Comparing HD3 with several state of the art DRL algorithms on four kinds of network environments (high, medium, low, oscillating) in live streaming scenarios. Results are normalized against the performance of HD3.

3.3.5 Results Analysis

Normalized QoE. Figure 3.6 shows the normalized reward of A3C, DQN and DDQN against HD3 under four network scenarios: high, medium, low, and oscillating. We have trained a single model with each DRL algorithm on a mixed dataset consisting of these four kinds network traces. In each network condition, every model is tested with 20 throughput traces and 3 video traces. It's clear that our proposed scheme HD3 can generate a single model to achieve the best performance in all four network conditions.

The reasons that HD3 performs better than others might come from two perspectives: the proposed dueling DQN based architecture can better handle variance in the mixed training data traces, and the comparing model can only output discretized latency limit values which makes it possible for the comparisons to miss the optimal latency limit values.

Hybrid Vs. Discretization. To better understand how different granularity levels of discretization of the latency limit value will influence the final performance, we conducted extensive experiments by uniformly discretizing the latency limit value range

$[0, 20]$ into 5 kinds of sets consisting of 5, 15, 25, 35, and 45 discrete values. To show why HD3 with hybrid action can outperform other models with only discrete actions, we also implemented the discrete version of HD3 (Dueling DQN) which can only output discrete actions.

Figure 3.7, Figure 3.8, Figure 3.9, and Figure 3.10 show the results of HD3 and other schemes on different granularity levels of discretization of latency limit value on high, medium, low, and oscillating bandwidth scenarios. From these figures, we can find that, when the granularity is in a low degree (5), then performance of the schemes is now high; when the degree of granularity becomes higher (from 5 to 15, 25 or 35), the performance may also become better. The reason for this is that, while the granularity level is higher, there will be more possible discrete latency limit values, thus making the models more possible to make better decisions. However, we can also find that, while the degree of granularity becomes too large (45), the performance will be worse. The reason for this phenomenon is that, when the granularity level of latency limit value is large, the total amount of possible candidate of action combinations will also become large, this will make it more difficult to train a model to converge to an optimal point or even a good point. The proposed scheme HD3 can outperform all the other schemes because it outputs a continuous value which makes it theoretically possible to find the optimal latency limit, and it does not increase the number of possible discrete action combinations which make it practically possible to converge to the optimal point or a relatively good point.

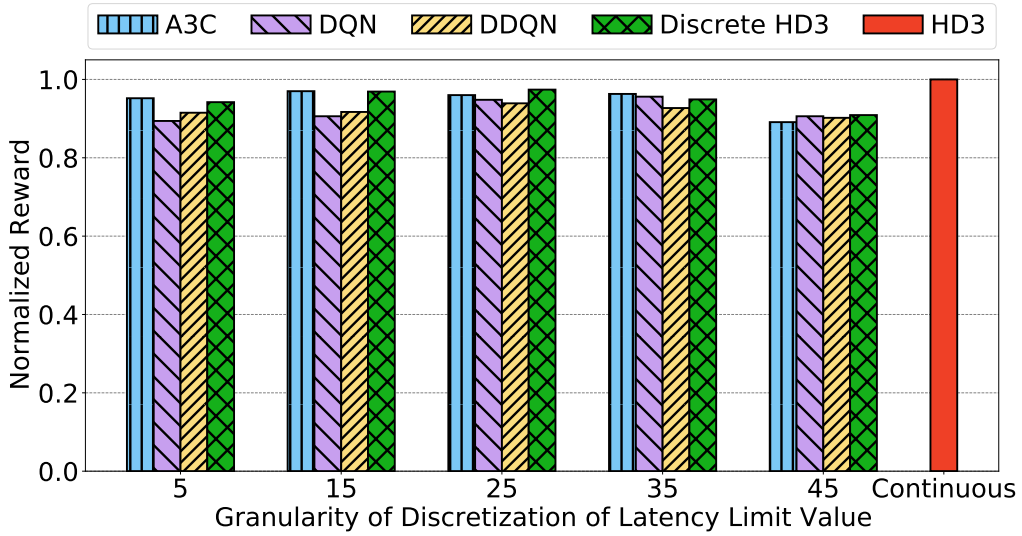


Figure 3.7: Analyzing performance of different DRL algorithms on different granularity of discretization of latency limit value on *High* bandwidth scenario. Results are normalized against the performance of HD3 which outputs a continuous latency limit value.

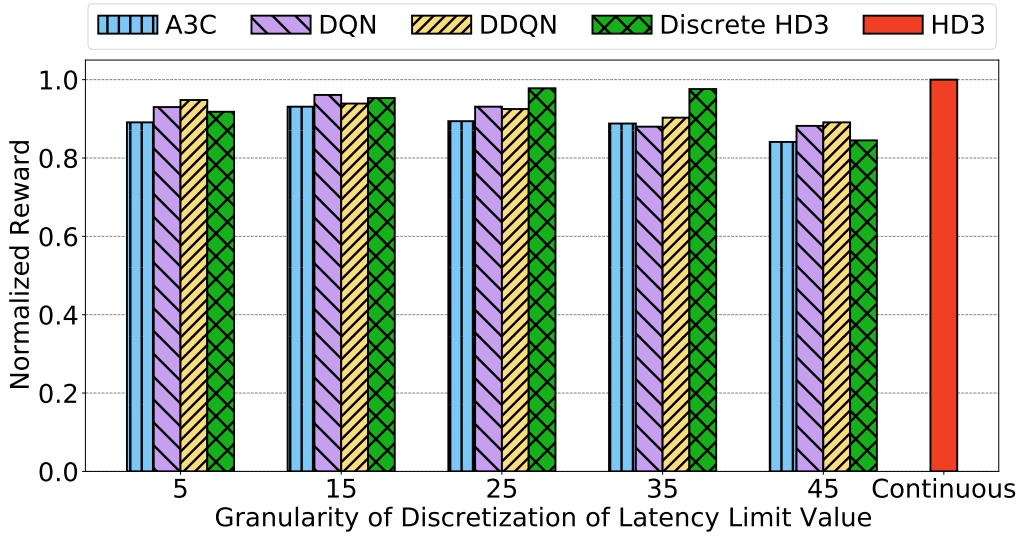


Figure 3.8: Analyzing performance of different DRL algorithms on different granularity of discretization of latency limit value on *Medium* bandwidth scenario. Results are normalized against the performance of HD3 which outputs a continuous latency limit value.

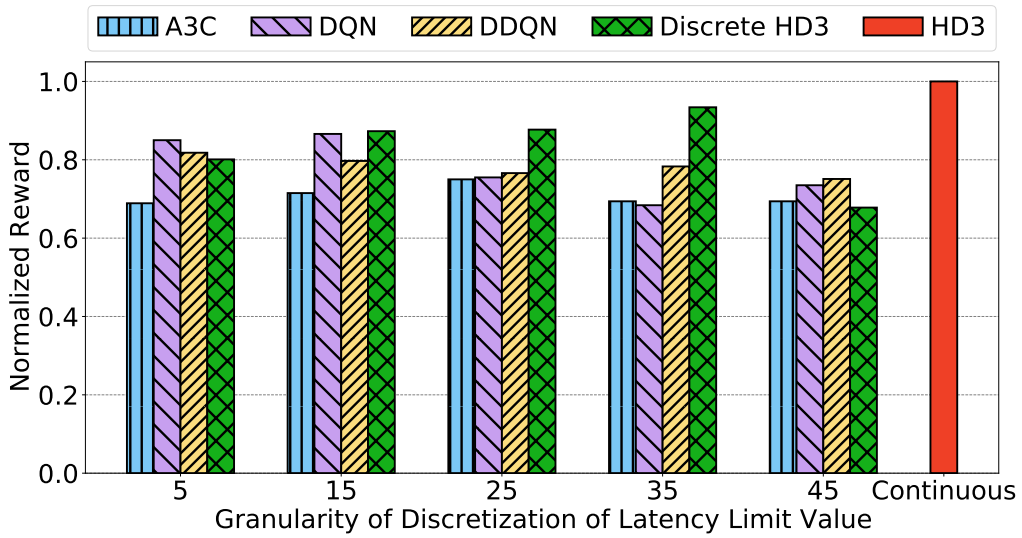


Figure 3.9: Analyzing performance of different DRL algorithms on different granularity of discretization of latency limit value on *Low* bandwidth scenario. Results are normalized against the performance of HD3 which outputs a continuous latency limit value.

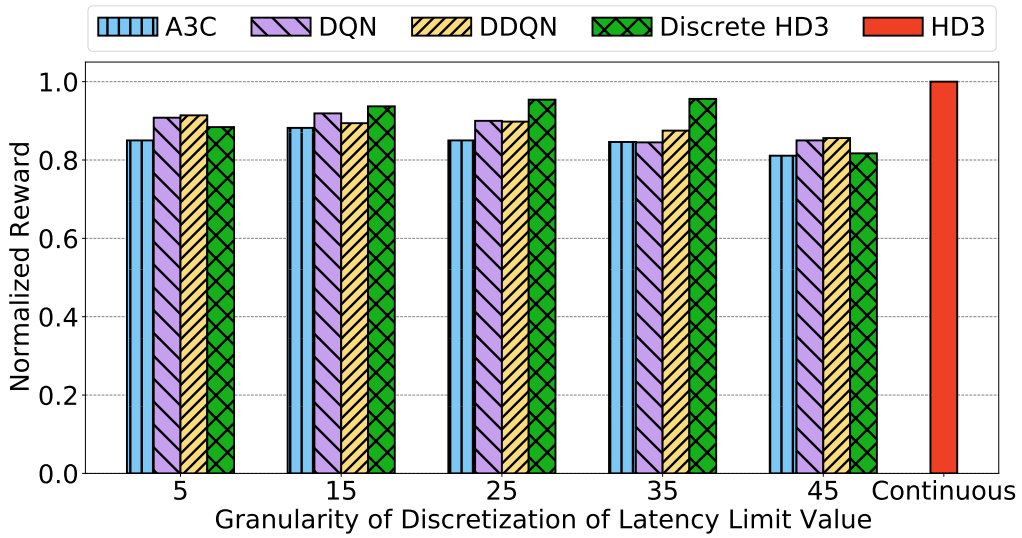


Figure 3.10: Analyzing performance of different DRL algorithms on different granularity of discretization of latency limit value on *Oscillating* bandwidth scenario. Results are normalized against the performance of HD3 which outputs a continuous latency limit value.

3.4 Summary

To address the challenges exposed in live video streaming scenarios, we proposed a novel DRL algorithm HD3, which can take discrete-continuous hybrid actions. Over four kinds of network environments and three kinds video streaming scenarios, we find that HD3 can generate a single model which can perform better than all the other comparing state-of-the-art DRL algorithms in all scenarios. HD3 can also be applied to solve other problems with hybrid action spaces.

4

Learning-based 360-Degree Video Streaming

4.1 Overview

In this chapter, we investigate tile-based viewport adaptive streaming for 360 videos, and propose the Plato system to leverage machine learning to address the inherent VPP and TBS issues. For VPP, Plato uses the long short term memory (LSTM) [71] based neural network model to predict users' future viewport orientation. Since the prediction errors of users' orientation can be harmful to QoE, we propose the concept of non-viewport to opportunistically cover actual areas that are slightly outside the predicted viewport areas. For TBS, Plato is equipped with a TBS agent that maps environment states (e.g., bandwidth, buffer size, tile sizes) to bitrate decisions for viewport and non-viewport areas to optimize the QoE, where the TBS agent is trained by the A3C algorithm [33] – a

state-of-the-art reinforcement learning algorithm [72].

To show how Plato performs with real data sets, we employ the real-world traces of user viewport and 4G bandwidth to train the neural networks, and our simulation results demonstrate that Plato outperforms existing schemes in terms of average QoE (including bitrate in viewport areas, rebuffering time, coefficient variation and smoothness).The contribution of this paper can be summarized as follows:

- We design a DRL architecture to determine tile bitrates for tile-based 360-degree video streaming.
- We propose the system – Plato, in which VPP and TBS agents are trained to predict viewport areas and select tile bitrates, respectively.
- We use real-world traces of viewport motions and network bandwidth, and our simulation results assert that Plato achieves significant improvement on QoE.

4.2 System Design

The architecture of Plato deployed within an HMD (worn by a 360 video player) is as depicted in Figure 4.1. There are several key components: state buffer, VPP and TBS agents. The state buffer collects information from the HMD and provides them to the agents. The VPP agent predicts future viewport based on the information of historical viewports from the state buffer. Considering potential viewport prediction errors, the VPP agent divides tiles into viewport, adjacent, and outside areas. In this way, the TBS agent can select high bitrates for actual viewport areas if the viewport prediction is not accurate (e.g. user's head is moving too fast). Finally, the TBS agent selects bitrate for each tile based on the state buffer information, including historical throughput, historical viewport prediction accuracy, and current buffer size. In the following, we are going to introduce how the VPP and TBS agents work in Plato.

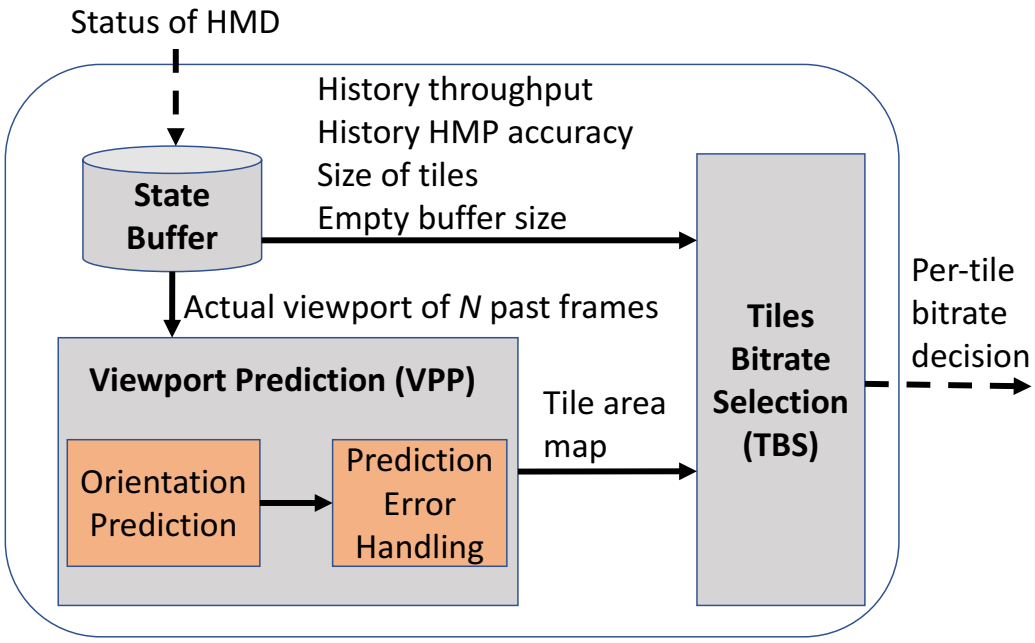


Figure 4.1: The architecture of Plato.

4.2.1 VPP – The Prediction of Viewport

In this part, we apply the LSTM based model to predict user's head orientation, and meanwhile discuss how we deal with the potential prediction errors.

Orientation prediction

We design a LSTM based model to predict future viewport, and illustrate its architecture in Figure 4.2. The LSTM layer takes the orientations $[X_{t-m+1}, ..., X_t]$ of m past video frames as inputs, where X_t is the array of $(yaw, pitch, roll)$ normalized to $[-1, 1]$. Then, the transform layer converts the output y of LSTM layer to normalized predicted orientation O_t. The equation of transform layer can be expressed as follows:

$$\tanh(Wy + b) = \frac{e^{Wy+b} - e^{-(Wy+b)}}{e^{Wy+b} + e^{-(Wy+b)}} \tag{4.1}$$

The network can predict future orientation of only one video frame now, but we can

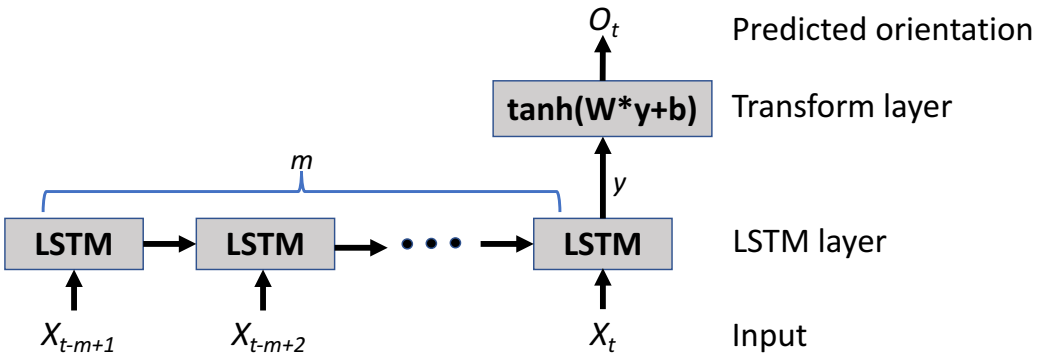


Figure 4.2: LSTM based predictor.

easily predict the next n orientations by making the last prediction as inputs. Then, we can calculate the viewport area according to the collected orientations in one segment length. Although the LSTM based model can predict at a relatively high accuracy, prediction errors may also take place, especially when the prediction window is long. Next, we will introduce how we handle the potential prediction error of $(yaw, pitch, roll)$.

Prediction error handling

We handle the potential prediction errors of head orientations by virtually enlarge the viewport range. In practice, the angles on a spherical surface can be defined by yaw and $pitch$ values (in degrees), both of which are discretized and mapped to pixels on a rectangular image with

$$x = \frac{width \cdot (yaw + 180)}{360}, \qquad y = \frac{height \cdot (90 - pitch)}{180}. \tag{4.2}$$

Here, we consider the size of the equirectangular image to be $width \cdot height$, the center of the equirectangular image is at $< yaw = 0, pitch = 0 >$, and the pitch angle increases in the upward direction.

We assume the standard vertical and horizontal FoVs of 90 and 110 degrees. We first mark the tiles covered by the actual FoV as viewport area. To handle the potential error we will mark the tiles covered by the FoVs from 90 to $90 + V$ degrees in vertical, and from 110 to $110 + H$ in horizon as the adjacent area, then all the remaining tiles will

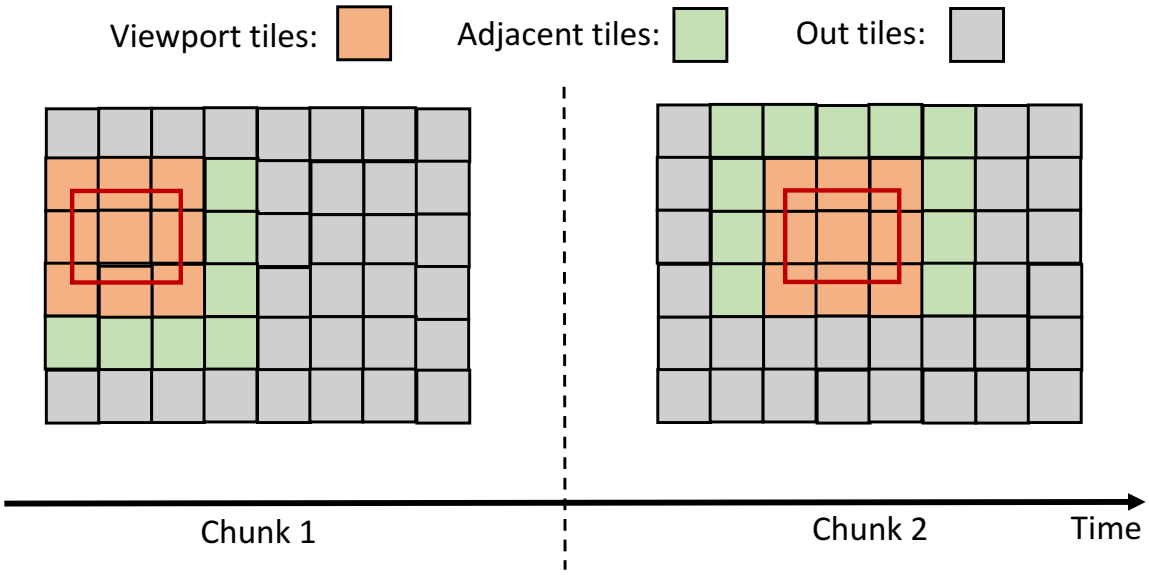


Figure 4.3: Division of tile areas.

be marked as outside area. The values of H, V should depend on the accuracy of the viewport predictor and the length of the prediction interval. Figure 4.3 illustrates the three areas: viewport area (yellow tiles), adjacent area (green tiles), and outside area (gray tiles). More detailed parameter settings are referred to section **??**. In the next part, we will introduce how we make the bitrate decisions for each area.

4.2.2 TBS – The Bitrate Selection of Tiles

Reinforcement Learning

In existing schemes, the TBS policy is typically generated by fixed heuristics, and our approach is to apply RL to generate it from observations. Figure 4.4 illustrates how the TBS agent interacts with a 360 video player. At each time step t, the TBS agent makes an action a_t according to its observed state s_t. After the action is made, the state transitions to s_{t+1} and the TBS agent will receive a reward r_t. The objective of RL is to maximize the expected cumulative discounted reward $\mathbb{E}\left[\sum_{t=0}^{n} \gamma^t r_t\right]$, where $\gamma \in (0, 1]$ is a discounting factor since the values of later time steps are less influential.

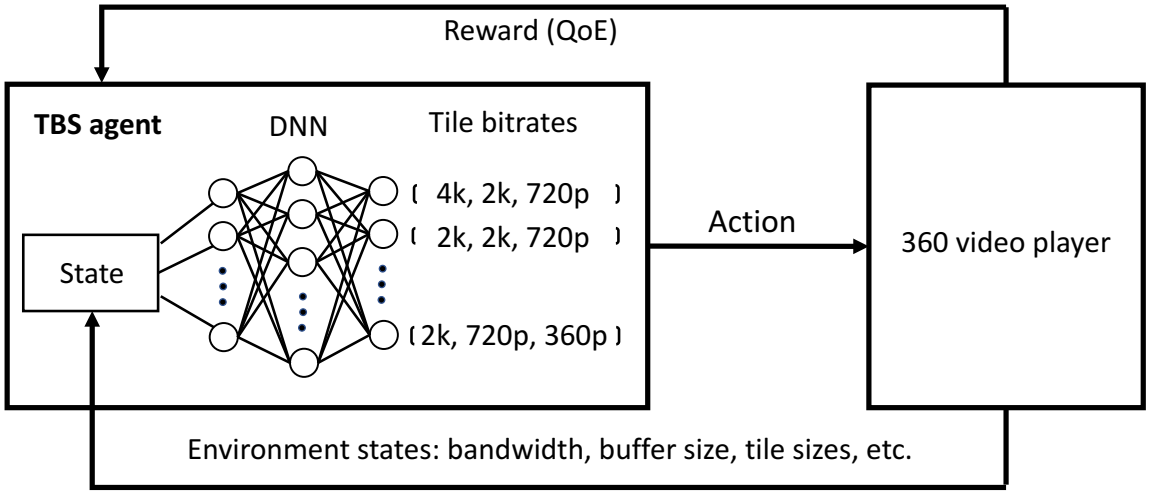


Figure 4.4: The workflows of RL.

In fact, the TBS policy can be interpreted as a neural network (see Figure 4.4). The TBS agent first acquires a set of state information (e.g., history bandwidth, current buffer size, next video tile sizes) from a 360 video player, feeds these values to policy (neural) network, and outputs bitrate selections for each tile area of the next chunk. Then, the environment collects the resulting QoE metrics and sends them to the TBS agent as a reward. With such a reward information, the TBS agent can then train and improve the policy neural network.

Training algorithm

In this section, our aim is to show how we train the TBS agent using RL. For this, we introduce the inputs needed in the training phase, explain the neural network architecture that we use for the TBS agent, and show how A3C trains the TBS agent.

Inputs: After the download of each chunk t, the TBS agent the state s_t as its input to its neural network, where

$$s_t = (\vec{x}_t, \vec{\tau}_t, \vec{vp}_m, \vec{ad}_m, \vec{out}_m, b_t, r_t, ratio_{vp}, ratio_{ad}, ratio_{out}). \tag{4.3}$$

For the past k 360 video chunks, the network throughput and the download time are

represented by $\vec{x_t}$ and $\vec{\tau_t}$, respectively. For the next 360 video chunk, $\vec{vp}_m$, $\vec{ad}_m$, $\vec{out}_m$ are the vectors of m available sizes for viewport, adjacent, and outside areas, respectively. b_t is the current buffer size and r_t is the average bitrate of the tiles among the actual viewport in the last downloaded video chunk. $ratio_{vp}$, $ratio_{ad}$ and $ratio_{out}$ refer to the percentage of actual viewport tiles that are predicted as viewport, adjacent, and outside tiles, respectively.

Policy: Given the state s_t, the TBS agent takes the action a_t (i.e., bitrate selections for three tile areas) in the next video chunk. In practice, each state may have more than one action choice. Therefore, the TBS agent makes its action decision based on the policy $\pi : (s_t, a_t) \rightarrow [0, 1]$. In other words, the TBS agent in state s_t chooses the action a_t with the probability $\pi(s_t, a_t)$. Here, we use a neural network to obtain the policy $\pi_\theta(s_t, a_t)$, where θ are the parameters of the neural network. The actor network in Figure 4.5 illustrates how Plato uses an NN to represent an TBS policy. The specific architecture of the neural network will be explained later. The critic network in Figure 4.5 is merely used to help train the actor network, we will introduce its function in next part.

Training Algorithm: Now, we are going to introduce the state-of-the-art actor-critic method A3C [33] as our training algorithm, which involves training of two neural networks. In the following, we provide detailed explanation of how A3C works.

Whenever the TBS agent downloads a video chunk, it will receive a reward r_t from the simulated environment. Since the goal of the TBS agent is to maximize the expected cumulative discounted reward, we set the reward based on the specific QoE metrics (see Sec. 4.3.5) to reflect the performance of TBS policy.

To maximize the expected cumulative discounted reward, A3C first needs to estimate the policy gradient of the expected total reward by observing the history state and action pairs. Given the policy network parameters θ, the gradient can be computed by

$$\nabla_\theta \mathbb{E}_{\pi_\theta}[\sum_{t=0}^{n} \gamma^t r_t] = \mathbb{E}_{\pi_\theta}[\nabla_\theta log\pi_\theta(s, a) A^{\pi_\theta}(s, a)], \tag{4.4}$$

where $A^\pi_\theta(s, a)$ is the advantage function, indicating the difference between the expected reward $v^{\pi_\theta}(s)$ for actions based on policy π_θ and the expected total reward when we

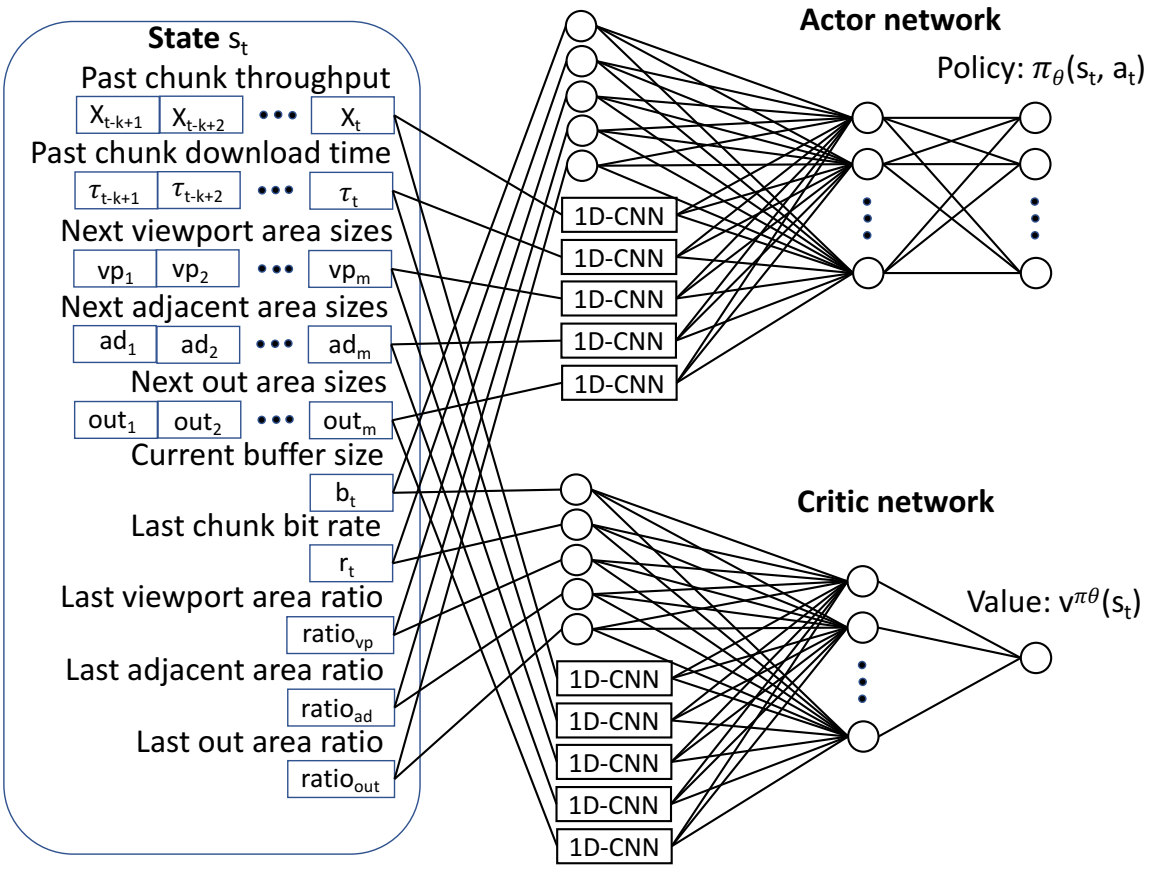


Figure 4.5: An illustration of A3C algorithm.

deterministically choose an action a in state s.

In practice, the agent will empirically estimate an unbiased $A^{\pi_\theta}(s_t, a_t)$ as $A(s_t, a_t)$ by sampling a trajectory of tile bitrate selection. The actor network parameters can then be updated with the learning rate α as follows:

$$\theta = \theta + \alpha \sum_t \nabla_\theta log\pi_\theta(s_t, a_t)A(s_t, a_t). \tag{4.5}$$

The output of the critic network is used to estimate the expected total reward $v^{\pi_\theta}(s)$, and A3C uses $v^{\pi_\theta}(s)$ to estimate the advantage $A(s_t, a_t)$. A3C updates the parameters of

the critic network as follows:

$$\theta_v = \theta_v - \alpha_v \sum (r_t + \gamma V^{\pi_\theta}(s_{t+1}; \theta_v) - V^{\pi_\theta}(s_t; \theta_v))^2, \tag{4.6}$$

where $V^{\pi_\theta}(\cdot; \theta_v)$ is the output by the critic network, α_v is the learning rate for the critic, r_t is the reward at time step t. The pseudocode of A3C can be found in [33]. More detailed parameter settings are referred to Sec. 4.3.1.

4.3 Evaluation

To evaluate the QoE performance of Plato, we develop a 360 video player simulator with real traces. We first introduce the parameter settings of Plato and real-world traces we used in the simulator and then give a detailed explanation of the 360 video player simulator. Next, we introduce the comparison schemes and QoE metrics. Finally, we show the simulation results and give some analysis.

4.3.1 Parameter Settings

We implement both LSTM based predictor and RL algorithm with pytorch [73], which is a popular open source machine learning framework. The detailed parameter settings in each part are as follows.

VPP settings. For LSTM based viewport predictor, we use one layer LSTM with hidden size of 128. For the error handling part, we set the values of H and V to 30 and 60, respectively.

TBS settings. For the actor network (see Figure 4.5), Plato passes $k = 8$ past throughputs, chunk download times to two identical 1D convolutional layers (CNN) with 128 filters, each of size 3 with stride 1. The sizes of next viewport, adjacent, outside areas are passed to other three 1D-CNNs with the same shape. The current buffer size, last chunk bitrate, the percentage of actual viewport tiles that are predicted as viewport, adjacent, and outside tiles are passed into five linear layers each with 128 neurons. The outputs from these layers are then passed to a linear layer with N neurons, where N is

the size of action space – the number of meta selections for the three tile areas. The critic network uses the same NN architecture, but its final output is only one neuron. During training, we set the discount factor $\gamma = 0.99$. The learning rates for the actor and the critic are both set to 0.0001.

4.3.2 Datasets

Video Preparation

A 360 video with the duration of 237 sec [74] is considered. The raw 3840×2016 video is extracted from the original clip and re-encoded using the kvazaar encoder [75], which is an open source software for H.265. The video is available in a 1-sec segment version, tiled as 12x6. Bitrate is in the range of [40, 16, 8, 5, 2.5, 1] Mbps according to the recommended settings from Youtube [76].

Viewport Motion

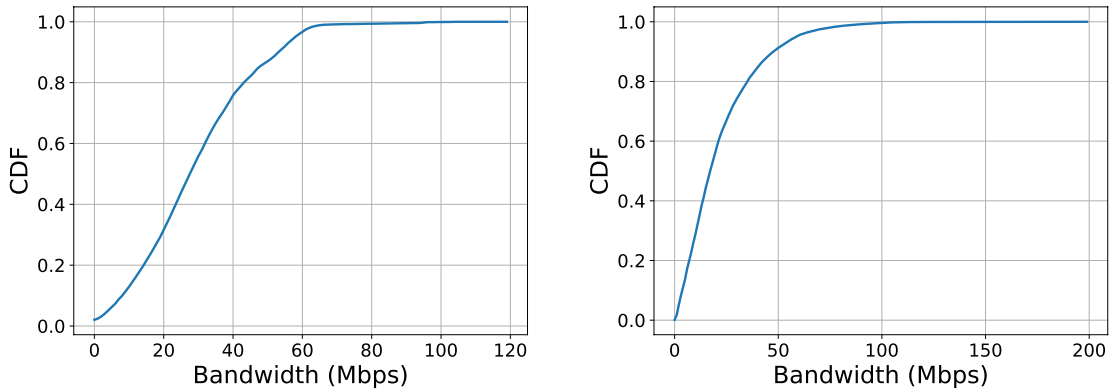
The dataset [74] includes data collected from 59 users watching video 70-sec 360 videos on the Razer OSVR HDK2 HMD. The selected videos span a wide range of 360 contents for which different viewers involvement (e.g., navigation patterns) could be expected.

The log files in the datasets record the unit quaternion[1] (q_0, q_1, q_2, q_3) of the HMD device in each timestamp. The unit quaternion (q_0, q_1, q_2, q_3) represents the rotations of an object in 3D space. For each object such as the HMD device in this coordinated system, the rotation vector $(yaw, pitch, roll)$ can be calculated from the unit quaternion as follows[2]:

$$\begin{bmatrix} yaw \\ pitch \\ roll \end{bmatrix} = \begin{bmatrix} \text{atan2}(2(q_3q_0 + q_1q_2), 1 - 2(q_0q_0 + q_1q_1)) \\ \text{asin}(2(q_2q_0 - q_3q_1)) \\ \text{atan2}(2(q_3q_2 + q_0q_1), 1 - 2(q_1q_1 + q_2q_2)) \end{bmatrix}. \tag{4.7}$$

We preprocess all the log files according to (4.7), and then randomly select 80% users' processed log files for all videos as the training dataset, and the remaining users' log files

[1] https://en.wikipedia.org/wiki/Quaternions_and_spatial_rotation
[2] https://stackoverflow.com/questions/5782658/extracting-yaw-from-a-quaternion



((a)) CDF of Belgium Bandwidth (test set)

((b)) CDF of Scaled Norway bandwidth (train set)

Figure 4.6: CDF of bandwidth

for all videos as the testing dataset. It's important to note that, the tests of VPP and TBS use the different viewport motion datasets.

Network Throughput

The test bandwidth dataset (Figure 4.6) we use contains the 4G networks traces along several routes in and around the city of Ghent, Belgium, from 2015-12-16 to 2016-02-04 [77]. All bandwidth logs are recorded based on the type of transportation: car, train, tram, bus, bicycle and foot. Given the high throughput observed (in some cases up to 95 Mbps) and the occasional switching to 3G, the logs have a relatively limited duration ranging from 166 to 758 sec. A total of 40 logs were collected, covering 5 hours of active monitoring.

[78] contains logs from TCP streaming session in Telenor's 3G/HSDPA mobile wireless network in Norway. In each test, video streams are downloaded at maximum speed. We reformatted the 3G dataset to have a 4G bandwidth by scaling the original bandwidth log file by a random number between $[10, 25]$, and we use the new cooked dataset as the training dataset.

4.3.3 360 Video Player Simulator Design

In this paper, we design a 360 video player simulator for training the TBS agent by RL and testing all the other comparison schemes. In fact, emulating the standard 360 video streaming is very slow, because the algorithm needs to wait until the chunks of a video is really downloaded before updating its neural network model. Therefore, we use a simulator to run the training algorithm alternatively. Next, we introduce the details of the simulator design.

We assume the client has the same behavior as HTTP/2, which can request all the tiles in a video segment at one time. We set the packet payload, RTT, and buffer size to 95%, 80 msec, 3 sec, respectively. In the simulator, we try to faithfully model the dynamics of 360 video streaming with real client applications. The simulator records the playback buffer usage information while streaming the 360 video. Before downloading, we will predict the future viewport with our proposed VPP agent using the real viewport motion traces. After the bitrate for each tile is decided, the simulator begins downloading the chunk. For each downloaded chunk, the simulator calculates the download time that is solely based on the chunk's size and the real-world network bandwidth traces. The simulator will then reduce the equal time video from playback buffer. The rebuffering events are carefully recorded while the buffer occupancy status changes, i.e., situations where download time is too long. In situations where the playback buffer is full, the simulator pauses requests for 500 msec before retrying[3]. After each chunk download, the simulator will calculate the real viewport from the input viewport motion traces, calculate the average bitrate of the tiles in the real viewport, and then pass the bitrate information and several other state observations to the RL agent for processing: the current buffer size, rebuffering time, chunk download time, size of the next tiles (at all bitrates), and the number of the remaining chunks in the video. Using this chunk-level simulator, Plato can "experience" 10 hours of video downloads in only 1 minute.

[3]This is the default request retry rate used by DASH players.

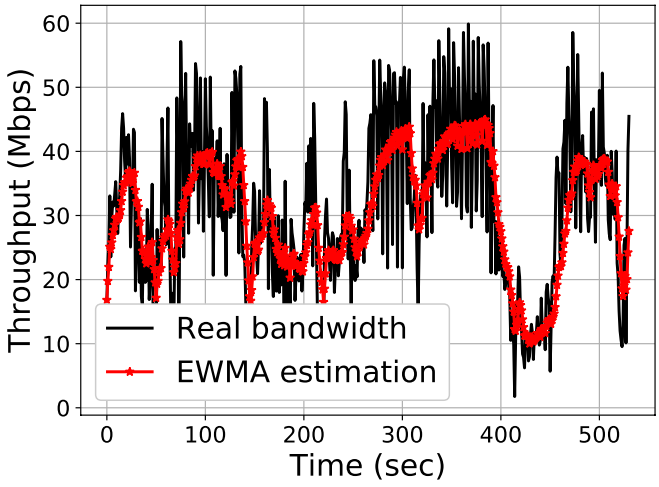


Figure 4.7: sample of 4G bandwidth

4.3.4 Comparison Metrics

- MM [43]: video chunks are split into tiles, and all tiles are logically divided into three areas: viewport, adjacent, and outside areas. According to their algorithm, the lowest quality is assigned to all the tiles of the video first. This initial allocation guarantees that all the tiles of the video are streamed to the user. Then, the available bandwidth budget is computed as the difference between the available bandwidth and the total bitrate allocated to the tiles. Next, the highest possible quality is assigned to the tiles, given the bandwidth budget, starting from viewport tiles, then adjacent tiles, finally outside tiles.

- FDA [44]: video chunks are split into tiles, and all tiles are logically divided into two parts: viewport and outside. The strategy is similar to MM, the lowest quality is assigned to all the tiles of the video first, then update the budget. Next assign the highest possible bitrate to tiles, starting from viewport tiles.

Standard throughput prediction method

Exponentially weighted moving average (EWMA) is used in Dash player and HLS player. EWMA is an exponentially weighted mean of previous throughput, as the following equation:

$$Estimate_{t+1} = \alpha Real_t + (1 - \alpha) Estimate_t \tag{4.8}$$

We reimplement the EWMA predictor as the industry Dash player [79] with the same parameter settings. In Figure 4.7, the red line shows the prediction performance of EWMA on one log file in the test dataset. We use EWMA for the comparison schemes MM and FDA.

4.3.5 QoE Metrics

Existing tile-based 360-degree video adaptive studies [29, 31, 45] have proposed the following QoE metrics to optimize:

1. Video quality: the quality of all tiles in the users' real viewport. Existing studies have utilized different metrics to reflect the quality of tiles, the authors in [31], [29], [45] have chosen average bitrate level, PSNR, and SSIM in their evaluation experiments respectively. These three metrics correlates positively with each other, that is, higher average bitrate level always means higher SSIM [45] and PSNR [80] values or vice versa. Here we choose average bitrate level rather than PSNR or SSIM, since it can be more easily calculated during both the training and evaluation phases.

2. Rebuffering time: the playback continuity which calculates the percentage of the duration of stall over the total video streaming .

3. Smoothness: average bitrate difference of viewport tiles between the adjacent time steps. Higher smoothness makes higher QoE.

4. Viewport spatial quality variance: If the quality of the content is not smooth, it will decrease the users' QoE, and we calculate this value according to the coefficient of variation (CV) of quality of content in the viewport.

4.3.6 Reward function design

To train a neural agent with DRL algorithms, we need to design a reward function first. During each training step, Following [20], we define a reward function based on the above four metrics as:

$$QoE = \sum_{n=1}^{N} q(R_n) - \mu \sum_{n=1}^{N} T_n - \lambda \sum_{n=1}^{N} cv - \sum_{n=1}^{N-1} |q(R_{n+1}) - q(R_n)| \tag{4.9}$$

For a video with N chunks, R_n is the average bitrate in the viewport of chunk n, and $q(R_n)$ maps the bitrate to the quality perceived by the users. T_n means the rebuffering time, while the third term penalizes video quality changes to favor smoothness. The final term also penalize the bitrate difference among tiles in the viewport to favor a consistent quality in a viewport.

The implementation of reward function plays a vital role on the performance of the neural agent trained by DRL. Specifically in our work, we need to design the quality function $q(R_n)$ and decide the values of coefficients μ, λ. The objective is to find an implementation which leads to an agent performing well on video quality, rebuffering and smoothness (and, ideally outperforming existing schemes in all dimensions). Considering giving the low bitrate with a low score and a high bitrate with a high score [20], we define the $q(R_n)$ as follows:

$$q(R_n) = \left\{ \begin{array}{ll} 1, & \text{for } 0 < R_n \leq 1.5 \\ 2, & \text{for } 1.5 < R_n \leq 3 \\ 3, & \text{for } 3 < R_n \leq 6 \\ 6, & \text{for } 6 < R_n \leq 10 \\ 9, & \text{for } 10 < R_n \leq 20 \\ 12, & \text{for } 20 < R_n \leq 40 \end{array} \right\}$$

We manually tuned the values of μ and λ based on the principle that optimizing one metric (by tuning the coefficients) diminishes the performance on other metrics (e.g. tuning up μ can reduce rebuffering time, meanwhile, result in lower video quality). The tuning process is basically a combination of random and binary search. We first evaluated

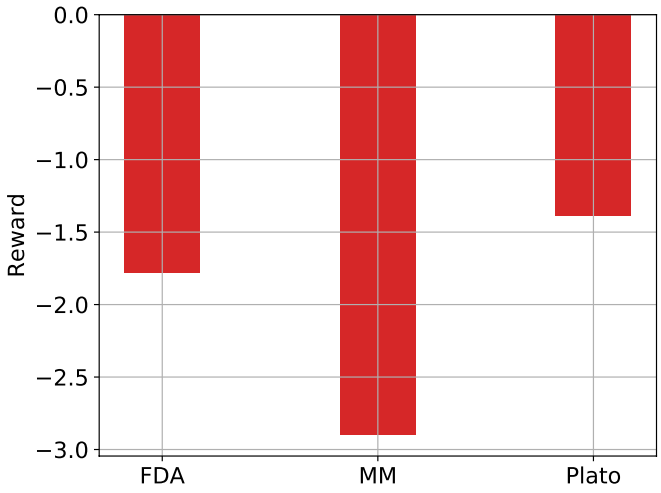


Figure 4.8: Average QoE

several randomly selected combinations of μ and λ to decide their upper and lower bounds, then utilize binary search algorithm to find a value of μ yielding high video quality with low penalty of rebuffering, and finally decide the value of λ in a similar way. In our simulation, μ and λ are set as as 43 and 5.3 respectively.

4.3.7 Result Analysis

Figure 4.8 shows the average QoE that each scheme achieves on our entire test datasets based on the QoE metric defined in 4.3.5. Since the penalty for rebuffering time, and CV is relatively large, the value the reward of all schemes are negative values. But we can also find that with considering all the four metrics together, our TBS agent can achieve the highest reward.

To better understand the QoE gains obtained by Plato, we analyzed Plato's performance on the individual terms in our general QoE definition. Specifically, Figure 4.9 compares Plato with the comparison schemes in terms of the the average bitrate in the actual viewport, the penalty from rebuffering, the coefficient of variance (CV) and smoothness. From the figure, we find that Plato can achieve the best average bitrate among all schemes, with less rebuffering time penalty than other two schemes. The reason is that our

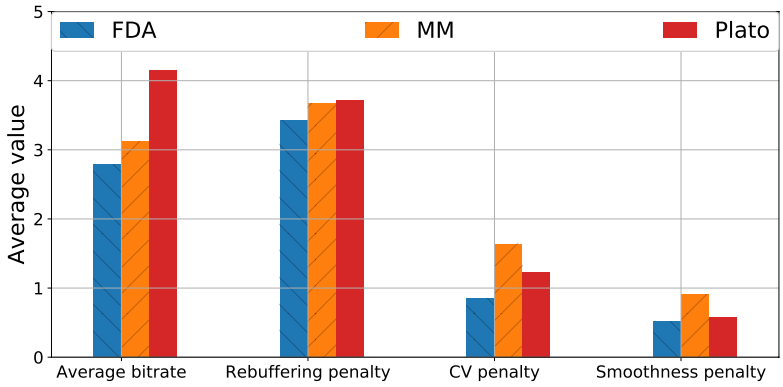


Figure 4.9: Comparing Plato with existing algorithms by analyzing their performance on the individual components in the general QoE definition

definition of QoE function favors high quality video, assigning the highest utility to the top three bitrates available for our test video. Since the penalty of rebuferring time is also large, Plato also does not receives much rebuffering time. Here, we can say that Plato can achieve much higher video quality and similar rebuferring time than other scheme. Plato also outperform MM in term of CV and smoothness, the reason is that by learning from a trace, the TBS agent can have some foresight to achieve less smoothness and low CV. FDA is better than Plato in terms of CV, because FDA only needs to select bitrate for two areas, but Plato need to select bitrate for three areas, this makes Plato have higher CV penalty.

It's important to note that the tile-based 360 video streaming is for achieving higher quality for the user under limited bandwidth. The results show that The learning based scheme can achieve much higher performance gain in terms of video quality, with less or similar rebuferring penalty, smoothness penalty, and CV penalty.

4.4 Summary

In practice, delivering entire 360 videos over the Internet could be prohibitive due to the limited available network bandwidth. To resolve this problem, we propose the Plato system to facilitate tile-based viewport adaptive streaming for 360 videos. In Plato, LSTM based neural network model is applied to predict users' future viewport orientation. To

be resilient to potential prediction errors, Plato delivers streams to part of non-viewport areas. In addition, Plato applies the A3C algorithm for training the TBS agent that maps environment states to bitrate decisions for both viewport and non-viewport areas. To compare Plato with existing schemes, we run simulations based on real traces of user viewport and 4G bandwidth usage, and our results demonstrate that Plato outperforms the comparison schemes in terms of various QoE metrics.

5 Sinusoidal Viewport Prediction for 360-Degree Video Streaming

5.1 Overview

The ever-increasing demand of immersive multimedia experience has stimulated content providers (e.g. YouTube and Vimeo) to roll out 360-degree video streaming. To fully embrace the panoramic and high-resolution multimedia experience, the unavailability of network bandwidth is an unsolved challenge. Streaming 360-degree videos requires unprecedentedly high bitrates when compared to video streaming of fixed viewing directions. Therefore, the ability to adaptively stream 360-degree videos in accordance with the dynamics of network bandwidth is decisive for its wide spread.

Commercialized 360-degree video streaming services mostly deliver entire 360-degree videos at constant quality. Since a user focuses on the so-called field of view (FoV) or

viewport of a sphere at a time, delivering the entirety of 360-degree video results in a waste of network bandwidth. Tile-based streaming[1] [29, 50, 81, 82] can resolve this issue by partitioning temporal video segments as spatially independent tiles and then associating viewport and non-viewport tiles with different qualities. Network bandwidth can thus be more efficiently utilized to facilitate 360-degree video streaming.

Smooth 360-degree video streaming requires a certain amount of video segments buffered for continuous playback. Existing solutions [29, 30, 34–36, 50–52, 55] suggest to pre-fetch all tiles of each segment, with tiles in the predicted viewport pre-fetched at a higher quality. Viewport prediction algorithms can be categorized into trajectory- and content-based methods as follows.

- Trajectory-based [29, 30, 36, 50–54, 83, 84]: existing methods predict future viewport based on a trajectory of either his own (single-user) or other users' (cross-user) historical rotations. Single-user methods predict future viewport based on user's past head rotation trajectory. Cross-user methods assume that different users have similar viewing behaviors on the same video, and determine the tile viewing probabilities of each individual user based on historical fixations of other users who have watched the same video. However, a trajectory-based prediction could be inaccurate due to its angle periodicity (e.g. -180° and 180° indicate the same direction). Moreover, cross-user algorithms cannot be deployed in live streaming scenarios which, by definition, lack historical information (FIGURE 5.1). Also, accuracy degrades with diversity in interest (viewing angle) for cross-user methods.

- Content-based [34, 35, 55–61]: existing algorithms typically use a saliency map [85] or a flow net [86] to extract image-features from the video content first, and then use a complicated model to predict future viewport with image-features and past rotations. Although content-based methods may yield a higher accuracy, the computational overhead of these algorithms exceeds the resources in practical deployments on mobile devices. Even if content-based algorithms can be deployed on the server, they may encounter scalability issues when there are millions of

[1] In practice, tile-based streaming has been standardized as part of dynamic adaptive streaming over HTTP (DASH) [3].

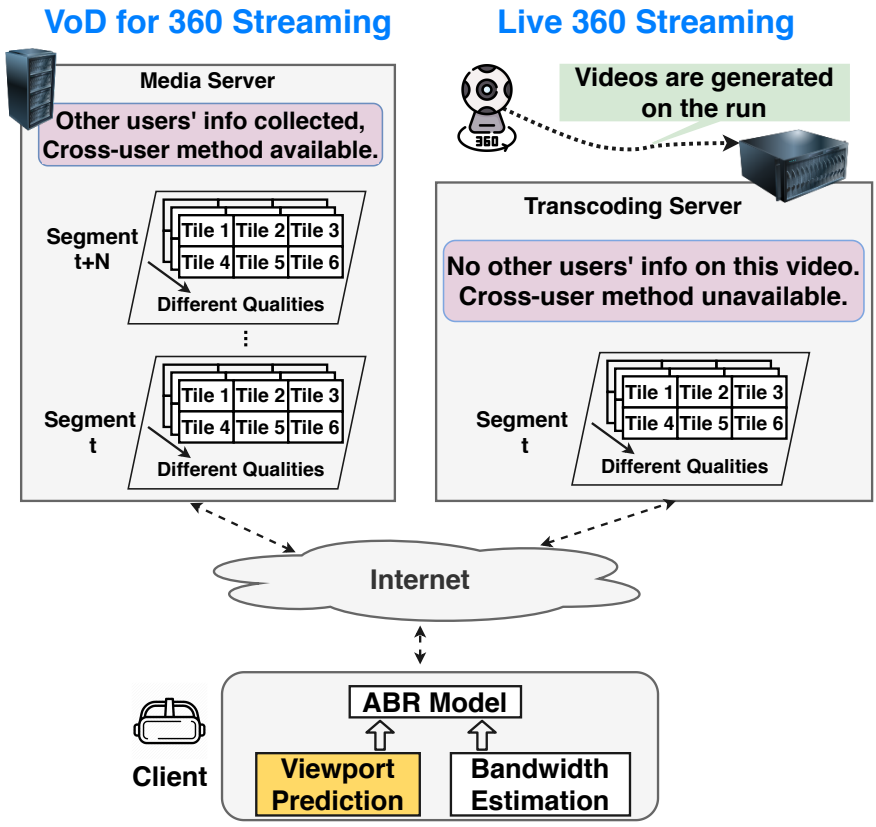
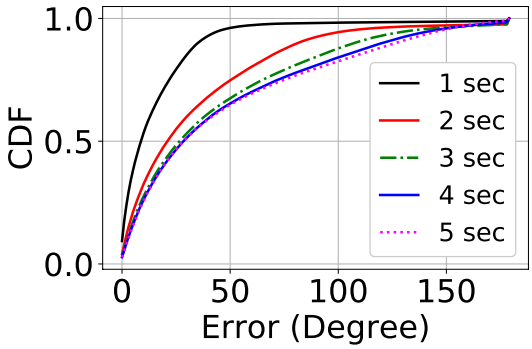


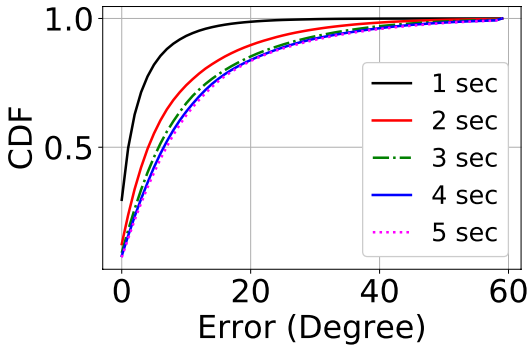
Figure 5.1: An illustration of tile-based streaming for 360-degree videos in VoD and live scenarios.

users watching millions of videos at the same time. In addition, VR video streaming systems already consume much computational power on encoding/decoding and rendering the 360 videos, a heavy viewport prediction model may make the whole experience worse than a lightweight model.

To address the above trajectory-based (inaccuracy and inapplicability to live streaming) and content-based (excessive computational overhead) concerns, we are motivated to improve the trajectory-based single-user method to build up a practical and accurate viewport prediction system applicable to both video on demand (VoD) and live streaming scenarios. To this end, we conduct data analysis on several datasets and find out that the angle periodicity issue on *yaw* (i.e. horizontal) direction can be avoided by converting



((a)) on *yaw* direction



((b)) on *pitch* direction

Figure 5.2: The prediction errors with respect to various time window lengths on the AV dataset (see Sec. 5.4.1).

degrees to the corresponding sinusoidal values.

Motivated by this observation, we design the sinusoidal viewport prediction (SVP) system for 360-degree video streaming in three stages: 1) orientation prediction, 2) error handling, and 3) tile probability normalization. First, we use linear regression (LR) least square method to predict the sinusoidal values of rotation angles. Next, we train a linear support vector regression (LinSVR) model to estimate the potential prediction errors to virtually enlarge the viewport area to cover more actual viewport tiles. Finally, we calculate the normalized viewing probability of tiles to improve adaptive bitrate (ABR) streaming performance. Overall, the contributions of this paper are:

- We identify that using sinusoidal values of rotation angles on *yaw* direction can improve the smoothness and linearity, thereby reducing prediction errors.

- We further improve the prediction accuracy by observing that head movement velocity and prediction time window positively correlate with prediction errors.

- We normalize viewing probabilities of tiles to improve the streaming performance of ABR.

5.2 Sinusoids versus Prediction Accuracy

Viewport prediction is critical in implementing tile-based 360 video streaming. Understanding why current models cannot predict future orientation accurately is important in order to design a better model. We conduct a study to understand the origin of the orientation prediction error, and propose a method to improve prediction accuracy.

5.2.1 Prediction Error Analysis

Figure 5.2 shows the CDF of the prediction error of both *yaw* and *pitch* for 1-5 secs. Prediction errors on the *yaw* axis are larger than towards *pitch* direction. One intuitive reason is that people move more in horizontal direction and remain steady in the vertical direction. Larger movement results in larger prediction error. Another reason is the angle periodicity issue (where $-180° = 180°$) in *yaw* direction. As shown in Figure 5.3, the black line (motion curve represented in degrees) shows a significant change when the head moves only a little, from slightly smaller than $180°$ to slightly larger than $-180°$ or inversely. A model trained by degrees may be inaccurate due to such a problem caused by the periodicity.

5.2.2 Conversion of Degrees to Sinusoid

Representing angles with Cosine and Sine avoids the periodicity issue. In Figure 5.3, observe that while user's head is moving between $180°$ and $-180°$, the cosine (red dashed line) and sine (blue dotted line) of the angle remain stable.

Figure 5.4 shows the predicted values of models trained with both the degrees and the sinusoidal values. Here, we use Frame 85 to 95 as the input to predict the future *yaw* degrees in the next 20 frames. We can find that model trained with sinusoidal values (red dashed line) can avoid the periodicity issue and get much less error than the model trained with degrees (green dotted line).

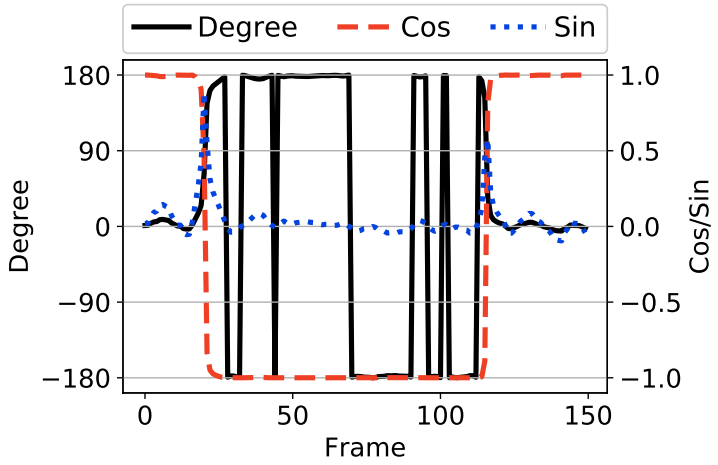


Figure 5.3: Head movement trace on *yaw* direction. The angle is represented by degree, and cosine/sine of the angle.

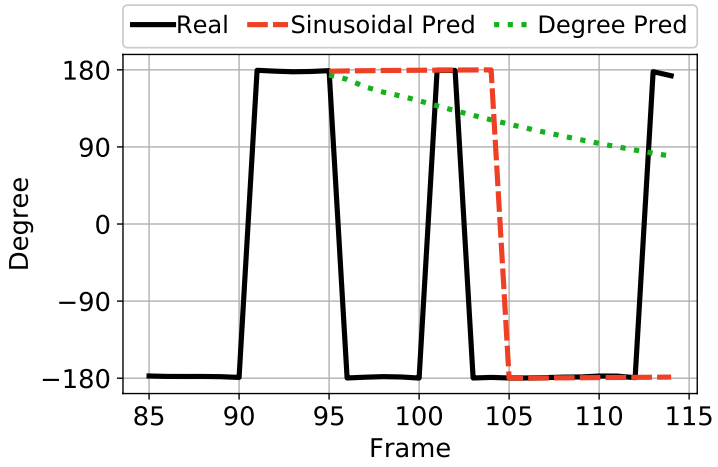


Figure 5.4: An illustration of predicted values of models trained with both the degrees and the sinusoidal values.

5.2.3 Why sinusoid is better?

Converting degree to sinusoidal values can render the trained model more accurate. Assume two metrics – smoothness and linearity to measure the data trace, because a smooth and linear curve can be more easily learned and results in a more accurate model.

Dataset	Metric	Degree	Cos	Sin
AV	Smoothness	10.34	21.21	20.04
	Linearity	0.069	0.197	0.115
THU	Smoothness	5.61	7.86	14.06
	Linearity	0.037	0.065	0.081
UT	Smoothness	7.27	8.30	13.98
	Linearity	0.036	0.087	0.0045

Table 5.1: Smoothness and linearity of degree, cosine and sine on *yaw* direction of the head motion datasets (see Sec. 5.4.1).

For a data trace $X = [x_1, ..., x_N]$, to obtain smoothness, we first derive the difference trace as $D = [d_1, ..., d_{N-1}]$; $d_i = x_{i+1} - x_i$; $\bar{d} = \frac{1}{N-1}\sum_{i=1}^{N-1} d_i$. We define the smoothness as the inverse of the standard deviation of D.

$$\text{Smoothness}(X) = \frac{1}{\sqrt{\frac{1}{N-1}\sum_{i=1}^{N-1}\left(d_i - \bar{d}\right)}}. \tag{5.1}$$

We use the absolute value of Pearson correlation coefficient between X and a linear vector $Y = [y_1, ..., y_N]$ as the linearity of X, that calculated as:

$$\text{Linearity}(X) = \frac{\sum_{i=1}^{N} |(x_i - \bar{x})(y_i - \bar{y})|}{\sqrt{\sum_{i=1}^{N}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{N}(y_i - \bar{y})^2}}, \tag{5.2}$$

where $\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$ and $\bar{y} = \frac{1}{N}\sum_{i=1}^{N} y_i$.

A larger absolute coefficient value relates to higher linearity, and a larger smoothness value corresponds to a smoother curve. Table 5.1 shows the smoothness and linearity of normalized degree, cosine, and sine on three head motion datasets. By representing degree with cosine and sine, the smoothness of data traces can increase to 99.5%, 95.4%, and 53.2%, while the linearity can improve to 126.1%, 97.3%, and 27.1%, for the AV, THU and UT datasets, respectively.

5.3 System Design

To incorporate the concept of sinusoidal viewport prediction into 360-degree video adaptive streaming, we propose the SVP system (as depicted in Figure 5.5) which consists of three stages: orientation prediction, error handling, and tile probability normalization. The SVP system first predicts future head rotations for each frame based on the collected historical head motion traces. Considering potential viewport prediction errors, the error handling module then divides tiles into viewport (VP), adjacent (AD) and outside (OUT) areas. Since the orientation prediction error is dependent on prediction time window length[2] and head movement velocity, we train a LinSVR [87] model to predict the orientation prediction error, thereby determining the size of adjacent areas. For each frame, we set a tile viewing probability for each tile, where VP tiles have the highest probability and OUT tiles get the lowest one. In the final stage, we normalize the tile probabilities in the next segment (containing M video frames) based on the viewing probabilities of tiles in the M frames. In the following, we are going to give more detailed descriptions of these three stages.

5.3.1 Orientation Prediction

The orientation prediction process is depicted on the left part in Figure 5.5. After collecting head rotations $[yaw_{t-N+1}, ..., yaw_t]$ and $[pitch_{t-N+1}, ..., pitch_t]$ of N past video frames, we convert them to cosine and sine vectors and then we apply linear regression (Least Square Method) to predict the cosine and sine values of both *yaw* and *pitch* in the next frame. Finally, we get *yaw* degree according to the equation below (same for *pitch*):

$$yaw_{t+1} = arctan2\left(\frac{\sin(y_{t+1})}{\cos(y_{t+1})}\right). \tag{5.3}$$

Although this model can only be used to predict the future orientation of one video frame from now, we can easily predict rotations in the next M frames by taking the last prediction results as inputs. Then, we can determine the viewport areas according to the

[2]The prediction time window length refers to the duration of playback time of video segments prefetched in the buffer.

collected orientations in one segment length. Even though this method can predict at relatively high accuracy, prediction errors may also take place, especially when the prediction time window is too long or the user's head movement is too fast. Therefore, in the following, we will introduce how we handle the potential prediction errors on both *yaw* and *pitch* directions.

5.3.2 Error Handling

We handle the potential prediction errors of head orientations by virtually enlarging the viewport range. In practice, the angles on a spherical surface can be defined by *yaw* and *pitch* values (in degrees), both of which are discretized and mapped to pixels on a rectangular image in accordance with

$$x = \frac{width \cdot (yaw + 180)}{360}, \quad y = \frac{height \cdot (90 - pitch)}{180}, \tag{5.4}$$

where *width* and *height* refer to the dimension of the equi-rectangular image, whose center is located at $(yaw, pitch) = (0, 0)$, and the *pitch* angle increases in the upward direction.

The middle part on Figure 5.5 illustrates the three areas: VP area (in yellow), AD area (in green), and OUT area (in gray). Consider that a typical setting of vertical and horizontal FoVs (i.e. red rectangles) are 90 and 110 degrees [50, 51, 88], respectively. We first mark the tiles covered by the actual FoV as the VP area. To handle the potential errors, we mark the tiles covered by the FoVs from 90 to $90 + V$ degrees on *pitch* direction, and from 110 to $110 + H$ on *yaw* direction as the AD area. Then, all the remaining tiles will be viewed as OUT areas.

In fact, the values of H and V are dependent on the accuracy of the orientation prediction. By testing the prediction model on a real-world dataset, as shown in Figure 5.6, we found that the prediction error is highly positively correlated with prediction window length and current head movement speed. For both *yaw* and *pitch* directions, the prediction error increases when the prediction window interval is longer or the motion velocity is larger. Based on this observation, we use the LinSVR model to estimate

the prediction error H (*yaw*) and V (*pitch*) based on head movement velocity and time window length as shown in Figure 5.5. The LinSVR model is a widely used method with enough high performance and low computational complexity.

Since we are trying to assign higher quality for viewport and adjacent areas and lower quality for outside area, we set the viewing probability for tiles in each area of per frame as follows: p_1 for VP tiles, p_2 for AD tiles, and p_3 for OUT tiles, with $p_1 > p_2 > p_3$. Next, we will introduce how to get the normalized tile viewing probability in one video segment which contains M frames.

5.3.3 Tile Probability Normalization

After error handling, the viewing probability of each tile in the next video segment is determined based on the tile probs in each frame. The right part in Figure 5.5 illustrates the viewing probability of one video segment in a 8×6 tilling manner as an example. We denote the viewing probability for the i-th tile in the k-th frame as v_i^k. Therefore, the total viewing probability for the i-th tile in the next video segment is calculated as:

$$\bar{v}_i = \sum_{k=1}^{M} v_i^k, \tag{5.5}$$

based on which the normalized viewing probability of the i-th tile can be obtained as

$$p_i = \frac{\bar{v}_i}{\sum_{i=1}^{M} \bar{v}_i}. \tag{5.6}$$

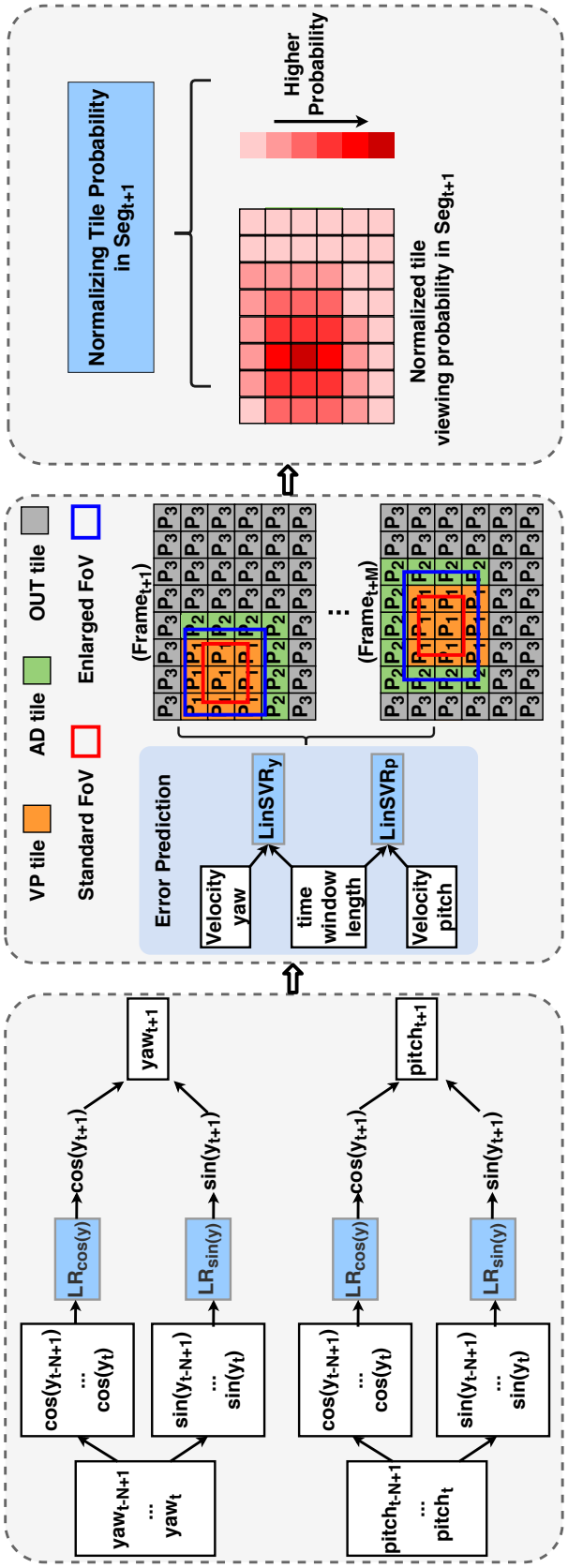
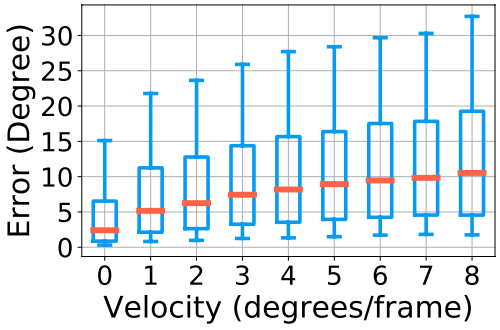
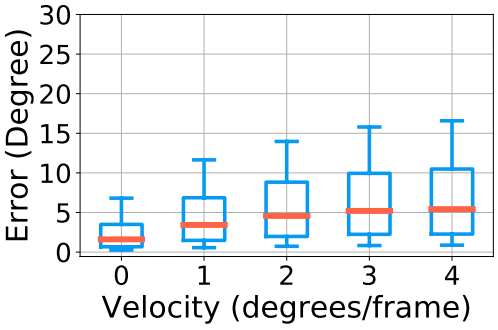


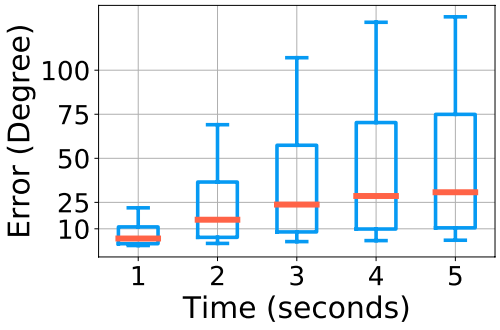
Figure 5.5: An illustration of the SVP system, which consists of three stages: orientation prediction, error handling, and tile probability normalization. The SVP system predicts based on head rotations collected by an HMD, and finally outputs viewing probability for each tile to the ABR model (corresponding to Figure 5.1).



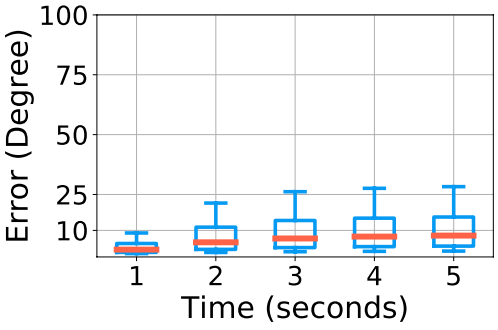
((a)) Velocity-Error (*yaw*)



((b)) Velocity-Error (*pitch*)



((c)) Time-Error (*yaw*)



((d)) Time-Error (*pitch*)

Figure 5.6: The relationship between prediction errors, time window lengths and head movement velocities in the AV dataset.

5.4 Evaluation

In this section, we evaluate the proposed viewport prediction system under various head motion datasets in terms of prediction error and accuracy. Then, we show how the proposed system can improve video quality under both fixed and dynamic bandwidth settings.

5.4.1 Simulation Settings

Experimental Setup

To validate user perceived quality under the proposed SVP system, we utilize a trace-driven emulation environment where a large corpus of network traces and viewport motion traces can be tested [51]. For each prediction, 10 past frames' rotations are used. In addition, the exact tile probabilities p_1, p_2 and p_3 for VP, AD and OUT tiles, respectively, should be obtained according to the design of ABR algorithms. In our experiments, we set p_1, p_2 and p_3 to 1.0, 0.5, 0.0, respectively, for simplicity.

Viewport Motion Datasets

We test the performance on three open viewport motion datasets. Different datasets are collected by different hardware, software and group of people and videos.

- AV [89]: 48 users watching 20 different entertaining omnidirectional videos on an HTC Vive HMD.
- THU [90]: A head tracking dataset composed of 48 users (24 males and 24 females) watching 18 sphere videos from 5 categories of contents.
- UT [91]: The authors gathered and produced 28 videos based on the taxonomy, and recorded viewport traces from 60 participants watching the videos.

As to the training and testing in our performance evaluation, we first evenly sample 10 frames per segment for all three datasets. To evaluate SVP's performance on unseen videos and users, we select 70% videos in the AV dataset for training, and remaining 30%

videos in the AV dataset and all videos in the dataset of THU and UT for testing. In the experiments of comparison scheme CLS, 90% of the users are randomly selected in each video for training, while the remaining 10% of users are used for testing.

Network Trace Dataset

The test bandwidth dataset we use contains the 4G networks traces along several routes in and around the city of Ghent, Belgium [77]. All bandwidth logs are recorded based on the types of transportation: car, train, tram, bus, bicycle and foot. A total of 40 logs were collected, covering a 5-hour active monitoring.

Video Dataset

A 360-degree video with a 237-sec duration [92] is employed for our simulations. The raw 3840×2016 video is extracted from the original clip and re-encoded using the kvazaar encoder [75], which is an open-source software for H.265. The video is available in a 1-sec segment version, tiled as 12x6 for each segment. According to the recommended settings [76], we consider six levels of qualities, corresponding to the bitrates of $\{1, 2.5, 5, 8, 16, 40\}$ Mbps.

Comparison schemes

- LR-G [93]: Linear regression model trained by gradient descent method.
- LSTM [51]: Long short term memory neural network, which is commonly used for time series prediction.
- MLP [94]: Multi-layer perception neural network.
- LR [50]: This tile-based streaming uses linear regression (Least Square Method) to predict future viewport, and then divides the tiles into three areas: viewport, adjacent, and outside areas. Each tile in the same area will be assigned the same quality [?].

- CLS [30]: group users with density-based spatial clustering of applications with noise (DBSCAN) in the server, then on the client end, classify the user to the corresponding cluster with a support vector machine (SVM) classifier, and finally obtain the viewing probability from the cluster.

- TJ [53]: first identify user clusters with a kind of spectral clustering algorithm, then fit a regression model for each cluster, finally predict with the regression model from the user's corresponding cluster.

- CUB [52]: first predict future fixations with LR, then utilize K-Nearest-Neighbor (KNN) to find the K nearest fixations of other users around the LR result to improve accuracy.

- LC [54]: first cluster users based on their quaternion rotations, then classify the target user to the corresponding cluster and estimate the future fixation as the cluster center. If no available cluster for the target user, then predict future fixation as the last sample.

5.4.2 Viewport Prediction Accuracy

Prediction Accuracy of Orientation

Here, we show how the prediction performance is improved by replacing degrees with their sinusoidal values on both *yaw* and *pitch* directions.

Figure 5.7 shows the mean prediction errors in *yaw* direction for 1- to 5-sec prediction time windows. First, prediction errors of all methods with sinusoidal values (i.e. in red bars) are lower than that with degrees (i.e. in blue bars) for all prediction time windows. In particular, we see that the prediction errors of LR-G, LSTM, MLP and LR with sinusoidal values (i.e. in red bars) can be reduced by 31%~47%, 13%~28%, and 5%~22%, in Figure 5.7(a), 5.7(b) and 5.7(c), respectively. This is because sinusoidal values can effectively avoid the periodicity issue on *yaw* direction. In addition, LR can always achieve the least prediction errors, which motivates us to design the orientation prediction part of the SVP system based on linear regression as in LR.

Prediction errors on *pitch* direction are shown on Figure 5.8. Since there is no periodicity issue on *pitch* direction, we cannot anticipate the performance improvement for all comparison schemes. However, we can observe that LR with sinusoidal values achieves the least prediction errors. Note that the prediction errors on *yaw* direction are more pronounced (than *pitch* direction), which requires more accurate viewport prediction designs in practice.

Prediction Accuracy of Tiles

We evaluate the tile prediction accuracy here and define tile prediction accuracy as the percentage of actual viewport tiles that are predicted as viewport tiles. As LR-G, LSTM and MLP yield much higher prediction errors than LR, we simply compare SVP with CLS, CUB, LC, and TJ in the following.

Figure 5.9 shows the average accuracy of these methods on three datasets in 1-, 2-, 3-, 4-, and 5-sec prediction time windows. Obviously, we see that, on all datasets, SVP outperforms CLS, CUB, LC and TJ in small time window lenghts (i.e. 1~3 sec) thanks to its accurate prediction of orientation. In larger time windows (i.e. 4~5 sec), the tile prediction accuracy of SVP reduces due to the accumulated prediction errors of orientation. This is because inaccurate viewport prediction of the previous frames inevitably leads to larger prediction errors of the next frame. Nevertheless, SVP remains comparable with respect to the cross-user methods CLS, CUB, LC and TJ. Note that the tile prediction accuracy of CLS is insensitive to time window lengths since CLS does not rely on previous frames for predicting the next frame.

5.4.3 Video Quality Assessment

To evaluate the proposed SVP system, we employ the target buffer-based bitrate selection algorithm in [29], where we modify the bitrate selection method by the following two steps: we first allocate the lowest quality to all tiles, and then assign the highest quality (within the estimated bandwidth budget) to tiles one by one in descending order of viewing probabilities.

In 360-degree video streaming, viewport prediction influences the bitrate allocation

of tiles and perceptual quality of contents in viewport. Since only a portion of each video is viewed by the user, the perceptual quality is decided by the tiles in the viewport. Therefore, we define the performance metric of *effective bitrate* as the sum bitrates of the tiles in the viewport.

Fixed Bandwidth Settings

Effective bitrates under fixed bandwidth settings. Figure 5.10 shows the CDF of an effective bitrate of different methods during streaming sessions for all three viewport motion trace datasets. Apparently, SVP yields higher effective bitrate than CLS, TJ, LC and CUB for all bandwidth settings as it achieves higher tile prediction accuracy. In particular, at the bandwidth setting of 30Mbps, SVP outperforms CLS, CUB, LC, and TJ by around 25.4%, 12.5%, 23.6% and 17.7%, respectively; at the bandwidth setting of 40Mbps, SVP outperforms CLS, CUB, LC, and TJ by around 13.6%, 4.0%, 12.9% and 15.7%, respectively. That is, the performance of SVP is more pronounced when the available bandwidth goes insufficient.

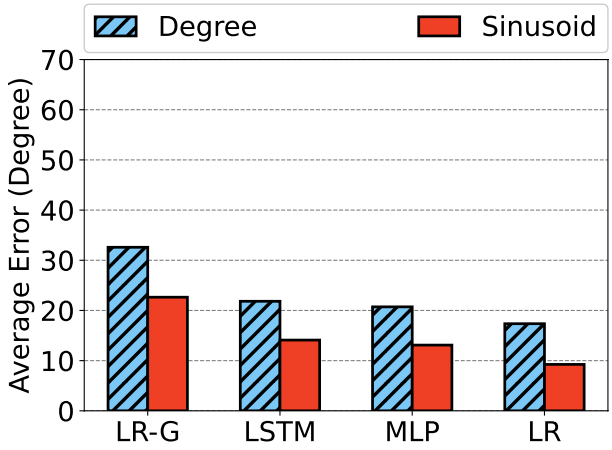
Dynamic Bandwidth Settings

To further evaluate the performance under various network conditions, we conduct a series of experiments under real-world network conditions. To make the simulation results repeatable, we employ the bandwidth trace of the 4G dataset collected at Belgium (see Sec. 5.4.1 for detailed description). In addition, we evaluate two buffer thresholds to see the variation of effective bitrates, where each buffer threshold serves as the upper limit of time window lengths.

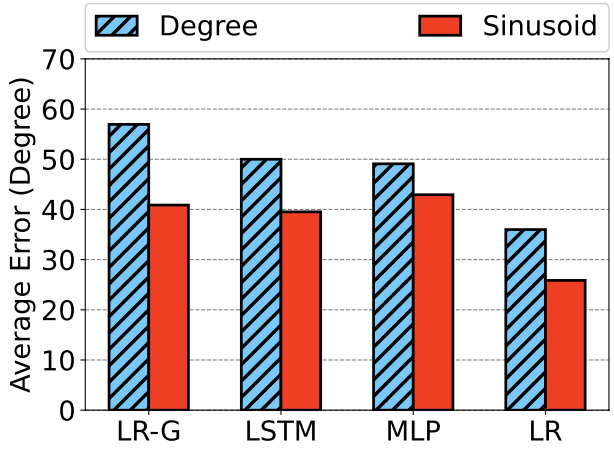
Figure 5.11 shows the CDF of effective bitrates of different methods on real-world network throughput with the 3-sec buffer threshold. Evidently, SVP performs better than CLS, CUB, LC, TJ in that it achieves higher tile prediction accuracy. Specifically, SVP can outperform CLS, CUB, LC, TJ by 8.6%~14.6%, 3.2%~10.7%, 4.7%~14.5%, and 10.0%~16.4% respectively, across the datasets (described in Sec. 5.4.1). On the other hand, when the buffer threshold goes larger (i.e. 6-sec), the prediction time window length will be longer. However, large buffer thresholds can result in inaccurate prediction, which reduces the

effective bitrate of SVP as well as its performance gain over the comparisons as shown in FIGURE 5.12. Since live and VoD 360-degree video streaming usually require small buffer thresholds, revealing the applicability of SVP in practice.

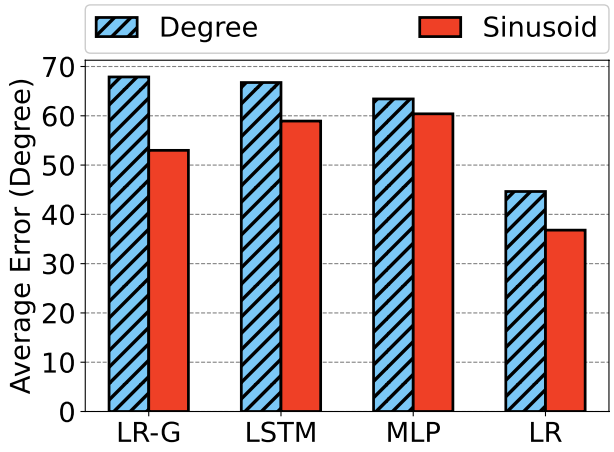
To see how much performance gain can be brought by the stages of the SVP system, we denote SVP-n as the SVP system from stage 1 to stage n, where $n \in \{1, 2, 3\}$. In Figure 5.13, we see that SPV-1 outperforms LR by 1.5%~13.1% thanks to the accurate orientation prediction. For SPV-2 and SPV-3, the performance gains will be further increased by 2.0%~4.8% and 2.8%~7.4%, respectively, revealing that the proposed error handling and tile probability normalization are both effective in elevating effective bitrates.



((a)) 1-sec time window

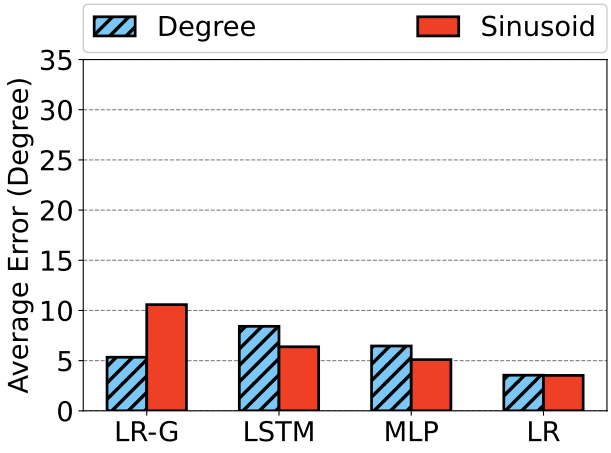


((b)) 3-sec time window

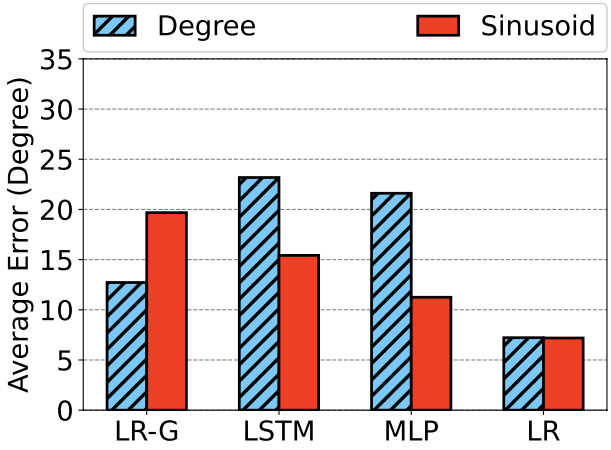


((c)) 5-sec time window

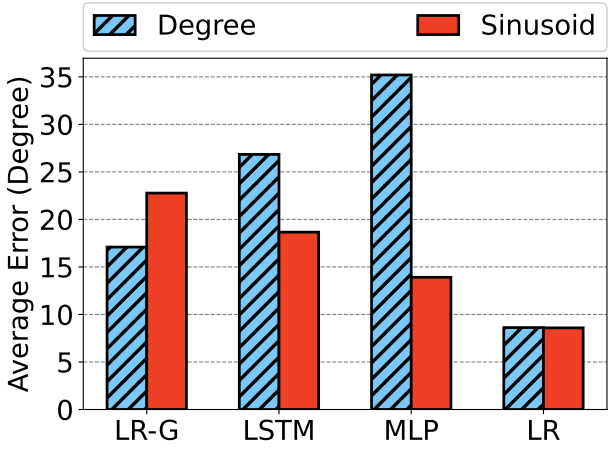
Figure 5.7: The prediction errors on *yaw* direction in 1-, 3-, 5-sec time window.



((a)) 1-sec time window

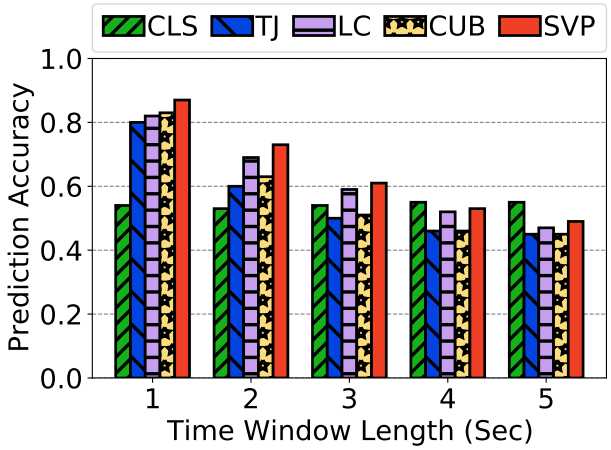


((b)) 3-sec time window

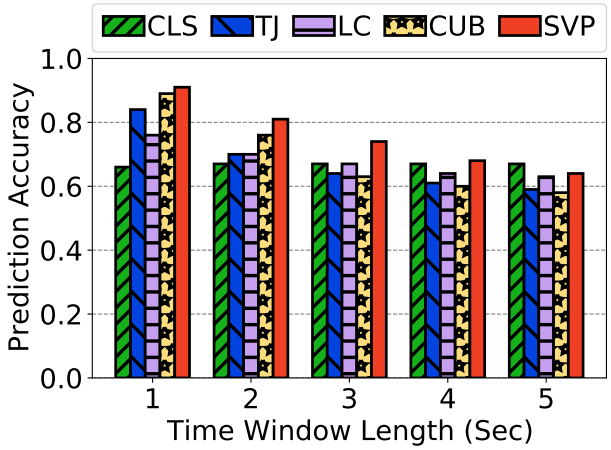


((c)) 5-sec time window

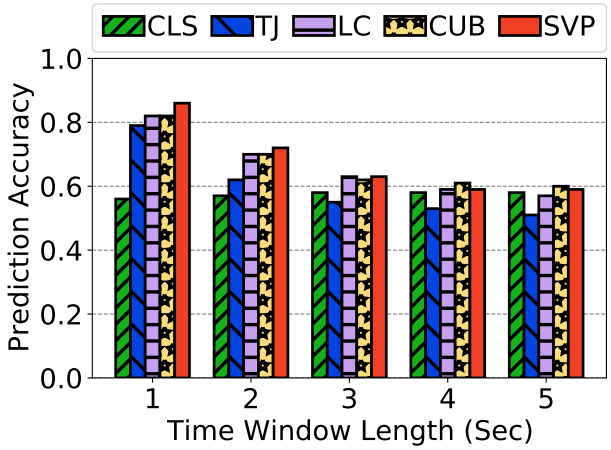
Figure 5.8: The prediction errors on *pitch* direction in 1-, 3-, 5-sec time window.



((a)) The AV dataset

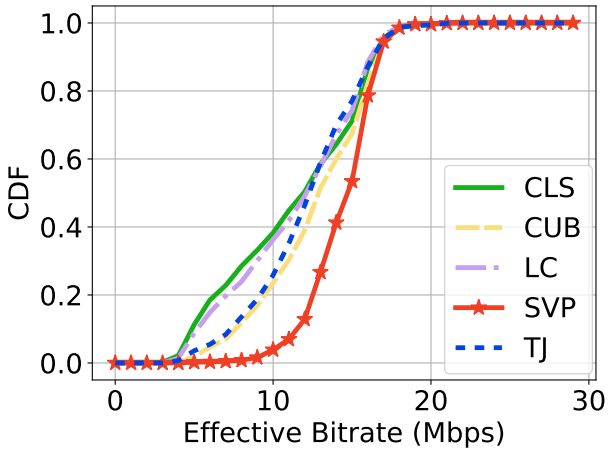


((b)) The THU dataset

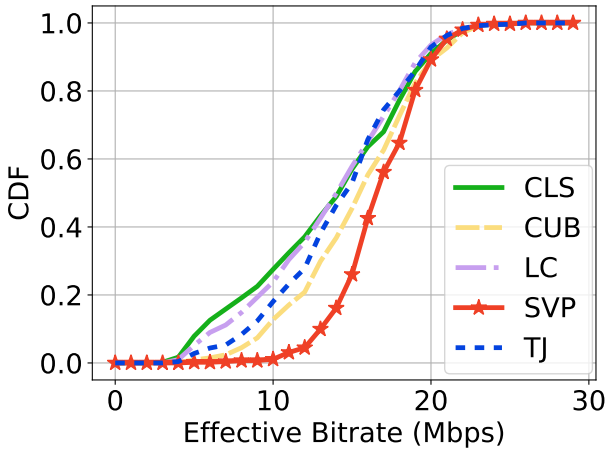


((c)) The UT dataset

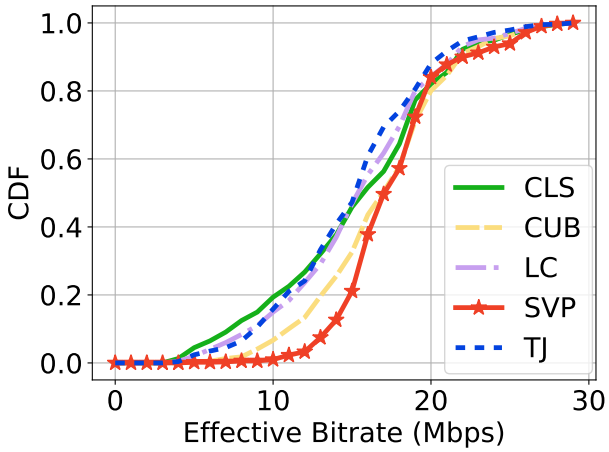
Figure 5.9: Tile prediction accuracy on different datasets.



((a)) Bandwidth = 30Mbps

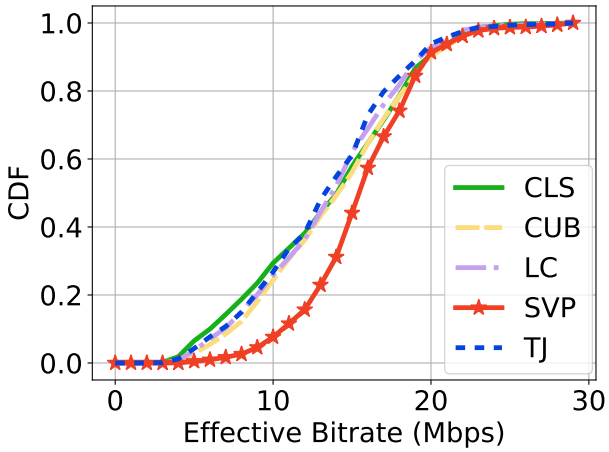


((b)) Bandwidth = 35Mbps

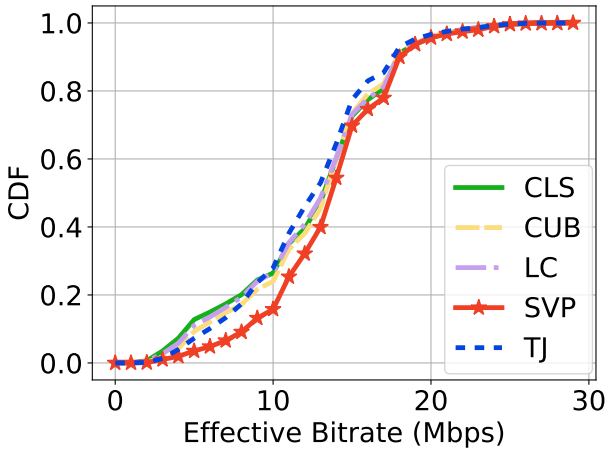


((c)) Bandwidth = 40Mbps

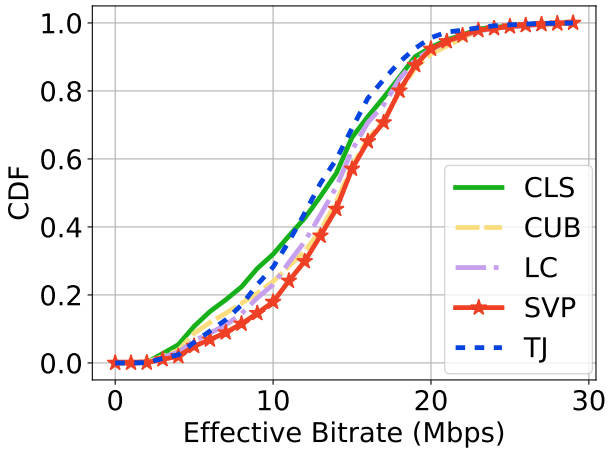
Figure 5.10: Effective bitrate comparison (fixed bandwidth, 3-sec buffer threshold).



((a)) The AV dataset

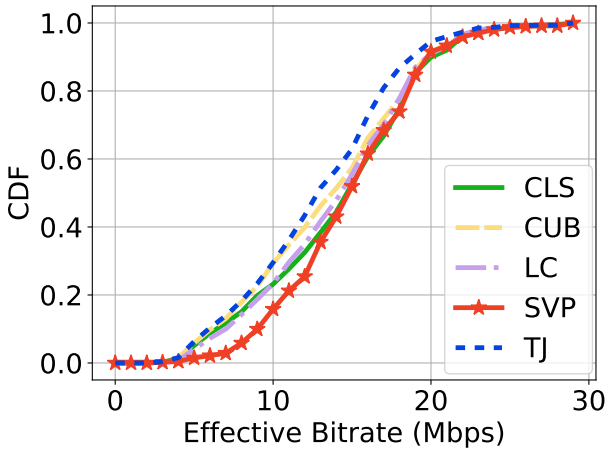


((b)) The THU dataset

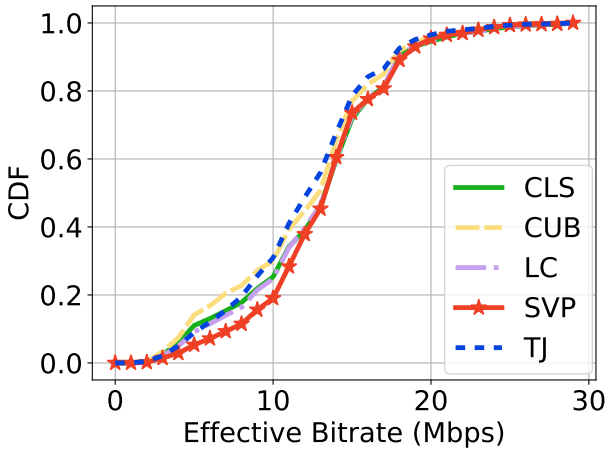


((c)) The UT dataset

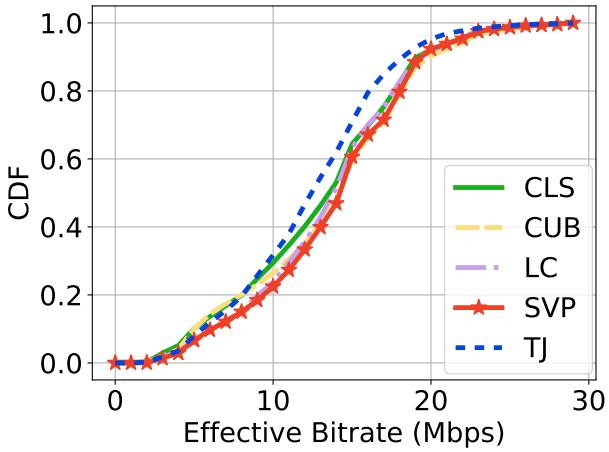
Figure 5.11: Effective bitrate comparison (dynamic bandwidth, 3-sec buffer threshold).



((a)) The AV dataset

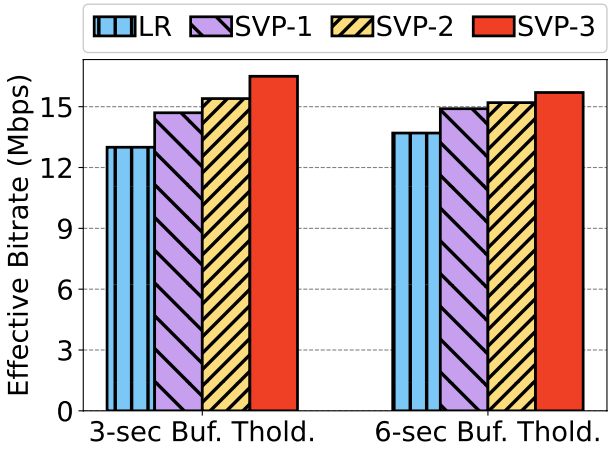


((b)) The THU dataset

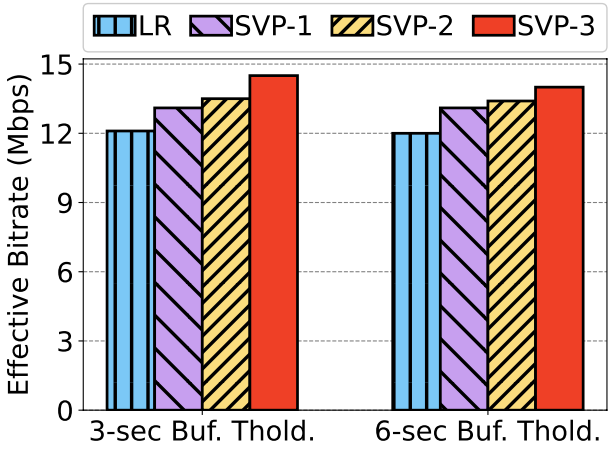


((c)) The UT dataset

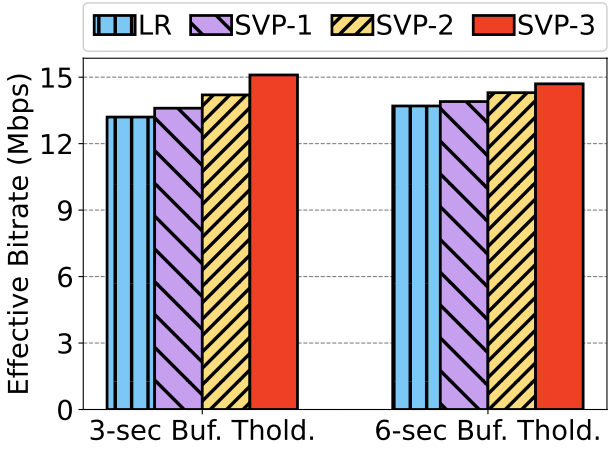
Figure 5.12: Effective bitrate comparison (dynamic bandwidth, 6-sec buffer threshold).



((a)) The AV dataset



((b)) The THU dataset



((c)) The UT dataset

Figure 5.13: Effective bitrate improvement by each stage of SVP (dynamic bandwidth, 3- and 6-sec buffer thresholds).

5.5 Summary

Tile-based streaming has emerged as a promising way to deliver high-quality 360-degree videos with limited bandwidth. In this paper, we propose the 3-stage SVP system to predict orientation, handle prediction errors and normalize viewing probabilities. By interpreting original rotations as sinusoidal values, the SVP system can effectively reduce prediction errors on *yaw* direction. We conduct simulations based on several real-world datasets, and our simulation results show that the SVP system can achieve better prediction accuracy and video quality than other comparison schemes under various buffer thresholds and bandwidth settings.

6 Conclusion

In this chapter, we first conclude the dissertation by summarizing its main contributions in Section 6.1, then discuss the limitations of our proposed schemes and the lessons we've learned in Section 6.2.1 and 6.2.2 respectively. Finally, Section 6.3 identifies the key future research topics that embrace learning-based approaches to challenges in the networking community.

6.1 Contributions

Our main contributions are developing learning-based systems to tackle challenges exposed by the live and 360-degree video streaming services. More specifically, we formulate the live streaming task as a reinforcement learning problem with discrete-continuous hybrid action spaces, then design a novel DRL algorithm-HD3 to solve it. The simulation results show that our scheme can generate a single neural agent which can

perform well under various network and video traces. Moreover, this distributed scheme can converge fast and be applied at scale in practice.

To address challenges exposed in the 360-degree video streaming scenarios, we develop two learning-based system for tile bitrate selection and viewport prediction respectively. We tackle the large action space issue in 360-degree video streaming, we divide the tiles into three classes, then apply a DRL algorithm to learn a neural agent to select bitrates for each tile class. We evaluate the proposed scheme-Plato in real-world traces of viewport motion and network bandwidth, and the results show that Plato achieves significant QoE improvement.

To more accurately predict future viewport, we propose a sinusoidal viewport prediction (SVP) system to overcome the periodicity issues on yaw direction. Moreover, we find that prediction errors positively correlate with head movement velocities and prediction time window, and utilize this finding to further improve the system performance. Simulation results demonstrate that the SVP system can achieve higher prediction accuracy thus leading to better video quality than comparison schemes.

6.2 Discussion

6.2.1 Limitations

This part examines the limitations of our work.

QoE metrics for 360 video streaming. There is no unique agreement on what is the best set of QoE metrics for 360-degree video streaming. In our work, we have chosen QoE metrics (also used in [29, 31, 45]) which can be easily obtained in simulations. However, the current chosen QoE metrics might not perfectly represent the subjective user QoE. Recently, some novel QoE metrics has been proposed. Yu *et al.* [32] presented 360JND-based PSPNR (360 Just Noticeable Difference based Peak Singla-to-Perceptible-Noise Ratio), which models the impact of viewport-moving speed, luminance change, and the difference in depth-of-field on the user perceived video quality. It is shown that 360JND-based PSPNR can estimate MOS more accurately than traditional video quality

metrics (e.g. PSNR). Marta *et al.* [25] has recently validated the application of another robust metric, VMAF, to 360 videos. Although, the QoE metrics we've chosen are able to show the efficiency of the proposed scheme to some degree, it will be more convincing to also conduct evaluations on these new metrics.

Reward function design. In current work, we have heuristically tuned the coefficient values of the reward function for 360-degree video streaming. However, we were not able to find an optimal value setting generating an model outperforming existing schemes in all dimensions. Our current heuristic-based searching method has missed many good combinations. To overcome this issue, a promising approach should be Bayesian optimization [95, 96]. Its general idea is to fit a function (usually a Gaussian process (GP) [97]) to map the evaluated combinations of coefficient values to the final QoE (video quality, rebuffering time, quality smoothness), and then sample the next test coefficient settings based on this function with the consideration of exploring unevaluated coefficients and exploiting the knowledge of best observed coefficient settings.

Simulator design. Another limitation comes from the simulator which is used for training the ABR agents. Although we have designed the simulators in segment level and carefully followed the same mechanisms from a real video player, it cannot perfectly emulate the system dynamics and influence of congestion control policy to the video streaming experience. It is worthy to build an simulation environment consisting of a frame (or packet) level video player simulator and a fully functioned network emulator.

Overhead, generality, interpretability of deep neural models. There are some limitations for learning-based approaches. Although the DRL trained neural agents can achieve better performance than heuristic-based schemes, the deep neural models usually consumes much more computational resources and power, thus leading to larger overhead. This makes it often unpractical to install deep neural models in mobile devices. Another common issue for learning-based methods is their generality. There is no strict guarantee that the trained models can still make good bitrate decisions on unseen video sequence and network conditions, or predict accurately on new users and videos. In

addition, the DRL trained neural agents are black-box whose mechanism cannot be fully understood, thus makes engineers difficult to correct the model when a wrong bitrate decision is made. To overcome these limitations, a promising way might be combining human heuristics and machine intelligence to design a robust, efficient and reliable system.

6.2.2 Lessons Learned

In this dissertation, we have developed learning-based systems for two decision making tasks and one prediction task. Here we summarize our lessons in terms of these two kinds of tasks.

Lessons from learning-based decision making. DRL algorithms are very promising for time series control problems. However, it's not naive to successfully apply DRL algorithms to solve real-world problems. Through our implementations of DRL-based system, we find that reward function design can significantly influence the final performance. Researchers should carefully tune the coefficients in the reward function to suitably handle the trade off among different QoE metrics. Other two hyper-parameters, learning rate and exploration rate, are also worth being noted by the engineers. Best learning rate and exploration rate should depend on the problem settings and datasets. Moreover, a distributed version of DRL converges much faster, which will allow the researchers to have more time to find good designs of reward functions and better hyper-parameter settings.

Lessons from learning-based prediction. Sometimes, utilizing more information or fancier neural network architectures may not increase the prediction accuracy for simple data traces. Moreover, this usually results in more complicated model which makes it unpractical to be deployed in mobile devices. To improve the prediction accuracy, another promising way should be analyzing the dataset and finding out the reasons for prediction errors, then utilizing feature engineering to overcome the issue.

6.3 Future Perspective

The application of learning-based approaches in networking systems is vast. There will be tremendous opportunities for machine learning techniques for replace human labor with machine intelligence. Next we give two broad research directions to design learning-based networking systems.

- Applying learning-based approaches to solve adaptive control problems in new networking applications.
- Integrating heuristics with learning for robust and efficient systems.

6.3.1 New networking challenges solvable by learning

Adaptive Volumetric Video Streaming Recent advances in the computing resources of machines have promoted interactive applications (e.g., virtual reality) to be viable service types to users. Most existing contents for these applications are 360-degree videos which provide users three degrees of freedom (3DoF) to freely rotate his/her head. However, people never stop their journey towards a more immersive, interactive, and expressive media. Recently, a new type of video, volumetric videos, which offer six degrees of freedom (6DoF) by allowing a viewer to freely change both the rotation (yaw, pitch, roll) and the position (X, Y, Z), have gained increased interest from the researcher community [98–104].

Volumetric videos are usually recorded by multiple cameras with depth sensors. Multiple cameras can capture all directions of a scene which allows the rotational freedom, and depth information of objects can be utilized to provide the transitional freedom. Volumetric videos can be represented by the technique of Point Cloud (PtCl) where each video frame contains multiple points or voxels. More specifically, each frame of volumetric videos contains a list of 3D points, and each point consists of its position and color information. This makes volumetric video streaming consumes much larger bandwidth than even 360-degree video streaming.

Viewport and rate adaptive approaches seems promising to overcome the aforementioned issue. However, it will be much more challenging to do viewport prediction and

rate adaptation in volumetric video streaming than in 360-degree video streaming. More specifically, in volumetric video streaming, both future rotation and positition values need to be predicted; and rate control needs to consider more information such as distances for each point cloud [99]. Besides these challenges, there lacks an agreed QoE function design for volumetric video streaming [100, 104], researchers still need to find appropriate QoE metrics to represent user-perceived quality while watching volumetric videos.

Learning-based approaches can be applied to solve the above challenges. To predict future head rotation and body position, various machine learning techniques can be utilized. Besides, DRL can still be considered as a promising approach to make rate decisions for different point clouds or part of the video on each adaptation step.

Adaptive Video Streaming for Analytics Unlike our previous work where videos are streamed for watching by human, videos can also be streamed for analyzing by machine. With the advances of computer vision techniques, many cities and companies have installed thousands of cameras in their streets or buildings for various video-analytic applications, such as face detection, traffic monitoring and control. In a typical traffic control application, cameras are installed along the street, and need to stream the recorded video frames back to the server for analysis over wireless links in real time. Then the deep neural networks (e.g., Yolo [105], VGG, [106]) deployed in the server conduct the object detection and classification tasks. To save bandwidth and reduce transmission latency, the applications need to re-size (change video resolution) and sample video frames [107]. However, a low resolution and sampling rate may decrease the inference accuracy. There are mainly two challenges for this kind of applications: conflicting QoE goals (latency, inference accuracy), and dynamic network conditions. To tackle these challenges, based on our lessons learned, it is very promising to leverage DRL to learn a neural agent to adaptively control the resolution and sampling rate of video frames.

6.3.2 Integrating heuristics with reinforcement learning

Most existing solutions for networking applications are designed either based on heuristics or machine learning techniques. Neither is perfect in practice, heuristic-based schemes

are usually simple but difficult to handle trade-off among multiple conflicting QoE goals and hard to adapt to various environment dynamics. While learning-based methods can often achieve better performance by overcoming the challenges of QoE conflicts and environment dynamics, they are difficult to be interpreted and their performance cannot be guaranteed for unseen (extreme) conditions during the training phases. Moreover, many reinforcement learning based systems are designed in a end-to-end way, which makes it impossible for the engineers to encode rarely happening cases directly into the system.

Recently, several works [108–113] have considered designing hybrid approaches by integrating heuristics (or physics) with learning to improve the performance. However, most of them focus on fusing heuristics with learning for prediction task, rather than decision making task which can be found in most networking applications. A promising way to fuse human heuristics with reinforcement learning for decision making, is to first design a framework containing several controlling parameters based on heuristics, then utilize reinforcement learning to learn a neural agent to adaptively control these parameters. In this case, all the rare happening cases and good heuristics in existing work can be manually encoded into the framework. Finally, we can get a system which can achieve high performance with high interpretability and robustness.

Bibliography

[1] 2018 Cisco Complete VNI Forecast and Trends Update. Available at https://www.cisco.com/c/dam/m/en_us/network-intelligence/service-provider/digital-transformation/knowledge-network-webinars/pdfs/1211_BUSINESS_SERVICES_CKN_PDF.pdf.

[2] Kjell Brunnström, Sergio Ariel Beker, Katrien De Moor, Ann Dooms, Sebastian Egger, Marie-Neige Garcia, Tobias Hossfeld, Satu Jumisko-Pyykkö, Christian Keimel, Mohamed-Chaker Larabi, et al. Qualinet white paper on definitions of quality of experience. 2013.

[3] Iraj Sodagar. The mpeg-dash standard for multimedia streaming over the internet. *IEEE MultiMedia*, 18(4):62–67, 2011.

[4] Ericsson Mobility Report. Available at https://www.ericsson.com/en/mobility-report/.

[5] Chang Ge, Ning Wang, Wei Koong Chai, and Hermann Hellwagner. Qoe-assured 4k http live streaming via transient segment holding at mobile edge. *IEEE Journal on Selected Areas in Communications*, 36(8):1816–1830, 2018.

[6] ACM Multimedia 2019 Grand Challenge-(Live Video Streaming). Available at https://www.aitrans.online/MMGC/.

[7] Huawei Virtual Reality/Augmented Reality White Paper. Available at http://www-file.huawei.com/-/media/CORPORATE/PDF/ilab/vr-ar-en.pdf.

[8] Stefano Petrangeli, Viswanathan Swaminathan, Mohammad Hosseini, and Filip De Turck. An http/2-based adaptive streaming framework for 360 virtual reality videos. In *Proceedings of the 2017 ACM on Multimedia Conference, MM 2017*, pages 306–314, 2017.

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

[10] Mohammad Hosseini and Viswanathan Swaminathan. Adaptive 360 vr video streaming: Divide and conquer. In *Multimedia (ISM), 2016 IEEE International Symposium on*, pages 107–110. IEEE, 2016.

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

[12] Tobias Hoßfeld, Raimund Schatz, Ernst Biersack, and Louis Plissonneau. Internet video delivery in youtube: From traffic measurements to quality of experience. In *Data Traffic Monitoring and Analysis*, pages 264–301. Springer, 2013.

[13] ITUT Rec. P. 10: Vocabulary for performance and quality of service, amendment 2: New definitions for inclusion in recommendation itu-t p. 10/g. 100. *Int. Telecomm. Union, Geneva*, 2008.

[14] Robert C Streijl, Stefan Winkler, and David S Hands. Mean opinion score (mos) revisited: methods and applications, limitations and alternatives. *Multimedia Systems*, 22(2):213–227, 2016.

[15] Athula Balachandran, Vyas Sekar, Aditya Akella, Srinivasan Seshan, Ion Stoica, and Hui Zhang. Developing a predictive model of quality of experience for internet video. *ACM SIGCOMM Computer Communication Review*, 43(4):339–350, 2013.

[16] M-N Garcia, Francesca De Simone, Samira Tavakoli, Nicolas Staelens, Sebastian Egger, Kjell Brunnström, and Alexander Raake. Quality of experience and http

adaptive streaming: A review of subjective studies. In *2014 Sixth International Workshop on Quality of Multimedia Experience (QoMEX)*, pages 141–146. IEEE, 2014.

[17] Florin Dobrian, Vyas Sekar, Asad Awan, Ion Stoica, Dilip Joseph, Aditya Ganjam, Jibin Zhan, and Hui Zhang. Understanding the impact of video quality on user engagement. *ACM SIGCOMM Computer Communication Review*, 41(4):362–373, 2011.

[18] Athula Balachandran, Vyas Sekar, Aditya Akella, Srinivasan Seshan, Ion Stoica, and Hui Zhang. A quest for an internet video quality-of-experience metric. In *Proceedings of the 11th ACM workshop on hot topics in networks*, pages 97–102, 2012.

[19] Michael Seufert, Sebastian Egger, Martin Slanina, Thomas Zinner, Tobias Hoßfeld, and Phuoc Tran-Gia. A survey on quality of experience of http adaptive streaming. *IEEE Communications Surveys & Tutorials*, 17(1):469–492, 2014.

[20] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. Neural adaptive video streaming with pensieve. In *ACM SIGCOMM 2017*, 2017.

[21] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. A control-theoretic approach for dynamic adaptive video streaming over http. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 325–338, 2015.

[22] Kevin Spiteri, Rahul Urgaonkar, and Ramesh K Sitaraman. Bola: Near-optimal bitrate adaptation for online videos. In *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*, pages 1–9. IEEE, 2016.

[23] Chun-Hsien Chou and Yun-Chin Li. A perceptually tuned subband image coder based on the measure of just-noticeable-distortion profile. *IEEE Transactions on circuits and systems for video technology*, 5(6):467–476, 1995.

[24] Zhou Wang, Ligang Lu, and Alan C Bovik. Video quality assessment based on structural distortion measurement. *Signal processing: Image communication*, 19(2):121–132, 2004.

[25] Marta Orduna, César Díaz, Lara Muñoz, Pablo Pérez, Ignacio Benito, and Narciso García. Video multimethod assessment fusion (vmaf) on 360vr contents. *IEEE Transactions on Consumer Electronics*, 66(1):22–31, 2019.

[26] Mingfu Li, Chien-Lin Yeh, and Shao-Yu Lu. Real-time qoe monitoring system for video streaming services with adaptive media playout. *International Journal of Digital Multimedia Broadcasting*, 2018, 2018.

[27] Konstantin Miller, Abdel-Karim Al-Tamimi, and Adam Wolisz. Qoe-based low-delay live streaming using throughput predictions. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 13(1):1–24, 2016.

[28] Yueshi Shen, Ivan Marcin, Josh Tabak, Abhinav Kapoor, Jorge Arturo Villatoro, and Jeff Li. Buffer reduction using frame dropping, July 3 2018. US Patent 10,015,224.

[29] Lan Xie, Zhimin Xu, Yixuan Ban, Xinggong Zhang, and Zongming Guo. 360probdash: Improving qoe of 360 video streaming using tile-based HTTP adaptive streaming. In *Proceedings of the 2017 ACM on Multimedia Conference, MM*, 2017.

[30] Lan Xie, Xinggong Zhang, and Zongming Guo. CLS: A cross-user learning based system for improving QoE in 360-degree video adaptive streaming. In *Proc. ACM MM*, Oct. 2018.

[31] Feng Qian, Bo Han, Qingyang Xiao, and Vijay Gopalakrishnan. Flare: Practical viewport-adaptive 360-degree video streaming for mobile devices. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, pages 99–114, 2018.

[32] Yu Guan, Chengyuan Zheng, Xinggong Zhang, Zongming Guo, and Junchen Jiang. Pano: Optimizing 360 video streaming with a better understanding of quality perception. In *Proc. ACM SIGCOMM*, Aug. 2019.

[33] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783, 2016.

[34] Yanyu Xu, Yanbing Dong, Junru Wu, Zhengzhong Sun, Zhiru Shi, Jingyi Yu, and Shenghua Gao. Gaze prediction in dynamic 360 immersive videos. In *Proc. IEEE CVPR*, Jun. 2018.

[35] Mengmi Zhang, Keng Teck Ma, Joo Hwee Lim, Qi Zhao, and Jiashi Feng. Deep future gaze: Gaze anticipation on egocentric videos using adversarial networks. In *Proc. IEEE CVPR*, Jul. 2017.

[36] Yuanxing Zhang, Yushuo Guan, Kaigui Bian, Yunxin Liu, Hu Tuo, Lingyang Song, and Xiaoming Li. EPASS360: QoE-aware 360-degree video streaming over mobile devices. *IEEE Trans. Mobile Comput.*, pages 1–1, 2020.

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

[38] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. In *Proceedings of the 2014 ACM conference on SIGCOMM*, pages 187–198, 2014.

[39] Gang Yi, Dan Yang, Abdelhak Bentaleb, Weihua Li, Yi Li, Kai Zheng, Jiangchuan Liu, Wei Tsang Ooi, and Yong Cui. The acm multimedia 2019 live video streaming grand challenge. In *Proceedings of the 27th ACM International Conference on Multimedia*, pages 2622–2626, 2019.

[40] Huan Peng, Yuan Zhang, Yongbei Yang, and Jinyao Yan. A hybrid control scheme for adaptive live streaming. In *Proceedings of the 27th ACM International Conference on Multimedia*, pages 2627–2631, 2019.

[41] Ruying Hong, Qiwei Shen, Lei Zhang, and Jing Wang. Continuous bitrate & latency control with deep reinforcement learning for live video streaming. In *Proceedings of the 27th ACM International Conference on Multimedia*, pages 2637–2641, 2019.

[42] Xiaolan Jiang and Yusheng Ji. Hd3: Distributed dueling dqn with discrete-continuous hybrid action spaces for live video streaming. In *Proceedings of the 27th ACM International Conference on Multimedia*, pages 2632–2636, 2019.

[43] Stefano Petrangeli, Viswanathan Swaminathan, Mohammad Hosseini, and Filip De Turck. An http/2-based adaptive streaming framework for 360 virtual reality videos. In *Proceedings of the 2017 ACM on Multimedia Conference*, pages 306–314. ACM, 2017.

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

[45] Jian He, Mubashir Adnan Qureshi, Lili Qiu, Jin Li, Feng Li, and Lei Han. Rubiks: Practical 360-degree streaming for smartphones. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, pages 482–494, 2018.

[46] Yuanxing Zhang, Yushuo Guan, Kaigui Bian, Yunxin Liu, Hu Tuo, Lingyang Song, and Xiaoming Li. EPASS360: QoE-aware 360-degree video streaming over mobile devices. *IEEE Trans. Mobile Comput.*, pages 1–1, 2020.

[47] Yuanxing Zhang, Pengyu Zhao, Kaigui Bian, Yunxin Liu, Lingyang Song, and Xiaoming Li. Drl360: 360-degree video streaming with deep reinforcement learning. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pages 1252–1260. IEEE, 2019.

[48] Tan Xu, Bo Han, and Feng Qian. Analyzing viewport prediction under different vr interactions. In *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*, pages 165–171, 2019.

[49] Bo Han, Yu Liu, and Feng Qian. Vivo: visibility-aware mobile volumetric video streaming. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, pages 1–13, 2020.

[50] Feng Qian, Lusheng Ji, Bo Han, and Vijay Gopalakrishnan. Optimizing 360 video delivery over cellular networks. In *Proc. ATC*, Oct. 2016.

[51] Xiaolan Jiang, Yi-Han Chiang, Yang Zhao, and Yusheng Ji. Plato: Learning-based adaptive streaming of 360-degree videos. In *Proc. IEEE LCN*, Oct. 2018.

[52] Yixuan Ban, Lan Xie, Zhimin Xu, Xinggong Zhang, Zongming Guo, and Yue Wang. Cub360: Exploiting cross-users behaviors for viewport prediction in 360 video adaptive streaming. In *2018 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6. IEEE, 2018.

[53] Stefano Petrangeli, Gwendal Simon, and Viswanathan Swaminathan. Trajectory-based viewport prediction for 360-degree virtual reality videos. In *2018 IEEE International Conference on Artificial Intelligence and Virtual Reality (AIVR)*, pages 157–160. IEEE, 2018.

[54] Afshin Taghavi Nasrabadi, Aliehsan Samiei, and Ravi Prakash. Viewport prediction for 360° videos: a clustering approach. In *Proceedings of the 30th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 34–39, 2020.

[55] Mai Xu, Yuhang Song, Jianyi Wang, MingLang Qiao, Liangyu Huo, and Zulin Wang. Predicting head movement in panoramic video: A deep reinforcement learning approach. *IEEE Trans. Pattern Anal. Mach. Intell.*, 41(11):2693–2708, 2018.

[56] Qin Yang, Junni Zou, Kexin Tang, Chenglin Li, and Hongkai Xiong. Single and sequential viewports prediction for 360-degree video streaming. In *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2019.

[57] Xiao Li, Siyi Wang, Chen Zhu, Li Song, Rong Xie, and Wenjun Zhang. Viewport prediction for panoramic video with multi-cnn. In *2019 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*, pages 1–6. IEEE, 2019.

[58] Xinwei Chen, Ali Taleb Zadeh Kasgari, and Walid Saad. Deep learning for content-based personalized viewport prediction of 360-degree vr videos. *IEEE Networking Letters*, 2(2):81–84, 2020.

[59] Xianglong Feng, Viswanathan Swaminathan, and Sheng Wei. Viewport prediction for live 360-degree mobile video streaming using user-content hybrid motion tracking. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 3(2):1–22, 2019.

[60] Xianglong Feng, Zeyang Bao, and Sheng Wei. Exploring cnn-based viewport prediction for live virtual reality streaming. In *2nd IEEE International Conference on Artificial Intelligence and Virtual Reality, AIVR 2019*, pages 183–186. Institute of Electrical and Electronics Engineers Inc., 2019.

[61] Xianglong Feng, Yao Liu, and Sheng Wei. Livedeep: Online viewport prediction for live virtual reality streaming using lifelong deep learning. In *2020 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pages 800–808. IEEE, 2020.

[62] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*, 2015.

[63] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[64] Jiechao Xiong, Qing Wang, Zhuoran Yang, Peng Sun, Lei Han, Yang Zheng, Haobo Fu, Tong Zhang, Ji Liu, and Han Liu. Parametrized deep q-networks learning: Reinforcement learning with discrete-continuous hybrid action space. *arXiv preprint arXiv:1810.06394*, 2018.

[65] Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado Van Hasselt, and David Silver. Distributed prioritized experience replay. *arXiv preprint arXiv:1803.00933*, 2018.

[66] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

[67] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. Neural adaptive video streaming with pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 197–210. ACM, 2017.

[68] Xiaolan Jiang, Yi-Han Chiang, Yang Zhao, and Yusheng Ji. Plato: Learning-based adaptive streaming of 360-degree videos. In *2018 IEEE 43rd Conference on Local Computer Networks (LCN)*, pages 393–400. IEEE, 2018.

[69] Tianchi Huang, Rui-Xiao Zhang, Chao Zhou, and Lifeng Sun. Qarc: Video quality aware rate control for real-time video streaming based on deep reinforcement learning. *arXiv preprint arXiv:1805.02482*, 2018.

[70] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[71] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[72] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[73] Pytorch. Available at https://pytorch.org/.

[74] Xavier Corbillon, Francesca De Simone, and Gwendal Simon. 360-degree video head movement dataset. In *ACM MMSys 2017*, number EPFL-CONF-227447, 2017.

[75] kvazaar: An open-source hevc encoder. Available at http://github.com/ultravideo/kvazaar.

[76] Recommended upload encoding settings, youtube. Available at https://support.google.com/youtube/answer/1722171?hl=en.

[77] Jeroen van der Hooft, Stefano Petrangeli, Tim Wauters, Rafael Huysegems, Patrice Rondao Alface, Tom Bostoen, and Filip De Turck. Http/2-based adaptive streaming of hevc video over 4g/lte networks. *IEEE Communications Letters*, 20(11):2177–2180, 2016.

[78] Haakon Riiser, Paul Vigmostad, Carsten Griwodz, and Pål Halvorsen. Commute path bandwidth traces from 3g networks: analysis and applications. In *Proceedings of the 4th ACM Multimedia Systems Conference*, pages 114–118. ACM, 2013.

[79] DASH Player Source Code. Available at https://github.com/Dash-Industry-Forum/dash.js/blob/development/src/streaming/rules/ThroughputHistory.js.

[80] Bruno Zatt, Marcelo Porto, Jacob Scharcanski, and Sergio Bampi. Gop structure adaptive to the video content for efficient h. 264/avc encoding. In *2010 IEEE International Conference on Image Processing*, pages 3053–3056. IEEE, 2010.

[81] Yu Guan, Chengyuan Zheng, Xinggong Zhang, Zongming Guo, and Junchen Jiang. Pano: Optimizing 360 video streaming with a better understanding of quality perception. In *Proc. ACM SIGCOMM*, Aug. 2019.

[82] Xavier Corbillon, Alisa Devlic, Gwendal Simon, and Jacob Chakareski. Optimal set of 360-degree videos for viewport-adaptive streaming. In *Proc. ACM MM*, Oct. 2017.

[83] Chenge Li, Weixi Zhang, Yong Liu, and Yao Wang. Very long term field of view prediction for 360-degree video streaming. In *2019 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR)*, pages 297–302. IEEE, 2019.

[84] Joris Heyse, Maria Torres Vega, Femke De Backere, and Filip De Turck. Contextual bandit learning-based viewport prediction for 360 video. In *2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pages 972–973. IEEE, 2019.

[85] Ziheng Zhang, Yanyu Xu, Jingyi Yu, and Shenghua Gao. Saliency detection in 360 videos. In *Proc. ECCV*, Sep. 2018.

[86] Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. FlowNet 2.0: Evolution of optical flow estimation with deep networks. In *Proc. IEEE CVPR*, Jul. 2017.

[87] Linearsvr: Linear support vector regression. Available at https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVR.html,

[88] Shahryar Afzal, Jiasi Chen, and KK Ramakrishnan. Characterization of 360-degree videos. In *Proceedings of the Workshop on Virtual Reality and Augmented Reality Network*, pages 1–6, 2017.

[89] Stephan Fremerey, Ashutosh Singla, Kay Meseberg, and Alexander Raake. AVtrack360: an open dataset and software recording people's head rotations watching 360° videos on an HMD. In *Proc. ACM MMSys*, Jun. 2018.

[90] Chenglei Wu, Zhihao Tan, Zhi Wang, and Shiqiang Yang. A dataset for exploring user behaviors in vr spherical video streaming. In *Proc. ACM MMSys*, Jun. 2017.

[91] Afshin Taghavi Nasrabadi, Aliehsan Samiei, Anahita Mahzari, Ryan P McMahan, Ravi Prakash, Mylène CQ Farias, and Marcelo M Carvalho. A taxonomy and dataset for 360° videos. In *Proc. ACM MMSys*, Jun. 2019.

[92] Xavier Corbillon, Francesca De Simone, and Gwendal Simon. 360-degree video head movement dataset. In *Proc. ACM MMSys*, Jun. 2017.

[93] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747 [cs.LG]*, Jun. 2017.

[94] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, Nov. 2016.

[95] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando De Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2015.

[96] Peter I Frazier. A tutorial on bayesian optimization. *arXiv preprint arXiv:1807.02811*, 2018.

[97] Carl Edward Rasmussen. Gaussian processes in machine learning. In *Summer School on Machine Learning*, pages 63–71. Springer, 2003.

[98] Mohammad Hosseini and Christian Timmerer. Dynamic adaptive point cloud streaming. In *Proceedings of the 23rd Packet Video Workshop*, pages 25–30, 2018.

[99] Jeroen van der Hooft, Tim Wauters, Filip De Turck, Christian Timmerer, and Hermann Hellwagner. Towards 6dof http adaptive streaming through point cloud compression. In *Proceedings of the 27th ACM International Conference on Multimedia*, pages 2405–2413, 2019.

[100] Feng Qian, Bo Han, Jarrell Pair, and Vijay Gopalakrishnan. Toward practical volumetric video streaming on commodity smartphones. In *Proceedings of the 20th International Workshop on Mobile Computing Systems and Applications*, pages 135–140, 2019.

[101] Alexander Clemm, Maria Torres Vega, Hemanth Kumar Ravuri, Tim Wauters, and Filip De Turck. Toward truly immersive holographic-type communication: Challenges and solutions. *IEEE Communications Magazine*, 58(1):93–99, 2020.

[102] Serhan Gül, Dimitri Podborski, Thomas Buchholz, Thomas Schierl, and Cornelius Hellge. Low latency volumetric video edge cloud streaming. *arXiv preprint arXiv:2001.06466*, 2020.

[103] Christian Timmerer and Ali C Begen. A journey towards fully immersive media access. In *Proceedings of the 27th ACM International Conference on Multimedia*, pages 2703–2705, 2019.

[104] Bo Han, Yu Liu, and Feng Qian. Vivo: visibility-aware mobile volumetric video streaming. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, pages 1–13, 2020.

[105] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[106] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[107] Junchen Jiang, Ganesh Ananthanarayanan, Peter Bodik, Siddhartha Sen, and Ion Stoica. Chameleon: scalable adaptation of video analytics. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 253–266, 2018.

[108] Elvis A Eugene, Xian Gao, and Alexander W Dowling. Learning and optimization with bayesian hybrid models. *arXiv preprint arXiv:1912.06269*, 2019.

[109] Jared Willard, Xiaowei Jia, Shaoming Xu, Michael Steinbach, and Vipin Kumar. Integrating physics-based modeling with machine learning: A survey. *arXiv preprint arXiv:2003.04919*, 2020.

[110] Prashanth Pillai, Anshul Kaushik, Shivanand Bhavikatti, Arjun Roy, and Virendra Kumar. A hybrid approach for fusing physics and data for failure prediction. *International Journal of Prognostics and Health Management*, 7(025):1–12, 2016.

[111] Alexander Y Sun, Bridget R Scanlon, Zizhan Zhang, David Walling, Soumendra N Bhanja, Abhijit Mukherjee, and Zhi Zhong. Combining physically based modeling and deep learning for fusing grace satellite data: Can we learn from mismatch? *Water Resources Research*, 55(2):1179–1195, 2019.

[112] Rahul Rai and Chandan K Sahu. Driven by data or derived through physics? a review of hybrid physics guided machine learning techniques with cyber-physical system (cps) focus. *IEEE Access*, 8:71050–71073, 2020.

[113] Tianchi Huang, Rui-Xiao Zhang, Xin Yao, Chenglei Wu, and Lifeng Sun. Being more effective and interpretable: Bridging the gap between heuristics and ai for

abr algorithms. In *Proceedings of the ACM SIGCOMM 2019 Conference Posters and Demos*, pages 12–14, 2019.