PARIMA: Viewport Adaptive 360-Degree Video Streaming

A thesis submitted in partial fulfillment of
the requirements for the degree of

Bachelor of Technology

in

Computer Science and Engineering

by

Sarthak Chakraborty
(Roll No. 16CS30044)

Under the guidance of
Dr. Sandip Chakraborty

Computer Science and Engineering
INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR
West Bengal, India
May, 2020

Certificate

*This is to certify that the work contained in this thesis entitled "**PARIMA: Viewport Adaptive 360-Degree Video Streaming**" is a bonafide work of **Sarthak Chakraborty (Roll no. 16CS30044)**, carried out in the Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur under my supervision and that it has not been submitted elsewhere for a degree.*

Dr. Sandip Chakraborty
Assistant Professor
Department of Computer Science & Engineering
Indian Institute of Technology Kharagpur,
West Bengal

May, 2020
Kharagpur

Acknowledgements

I would like to thank my guide, Dr. Sandip Chakraborty for his exceptional guidance and support, without which this project would not have been possible. He has always motivated me to explore as much as I can, look into multiple papers and try as many ideas as I can. He has always supported me through whatever problems I faced during the project.

I am grateful to my project partner, Lovish Chopra, for helping me and supporting me throughout the project, without whom it would have been really difficult to implement the project.

I express my deep gratitude to Mr. Abhijit Mondal for his guidance during the implementation of the project. I would also like to thank Mr. Rohit Verma, Ms. Snigdha Das, and Mr. Arani Bhattacharya (KTH Royal Institute of Technology) for providing their valuable inputs and helping us in the project.

In conclusion, I recognize that this project would not have been possible without the support from the Department of Computer Science and Engineering, IIT Kharagpur. Many thanks to all those who made this project possible.

Sarthak Chakraborty

Abstract

With increasing advancements in technologies for capturing 360° videos like omnidirectional cameras and VR headsets, advancement in streaming of such videos has grown to be a popular research topic. The high bandwidth and short response time requirement of streaming 360° videos has escalated the need of developing optimised streaming algorithms. Researchers have proposed various methods to tackle the problem, taking into consideration network bandwidth and viewport prediction. However, most of the existing work has been done without considering the video contents.

This report presents a method which uses the previous set of viewports along with the video contents, involving the trajectories of prime objects, to predict the next viewport. We claim that the head movement of a user majorly depends upon the trajectories of the prime objects in the video. The viewport prediction model uses the set of previous viewports and the object trajectories to obtain the predicted next set of viewports using a fast and efficient online learning algorithm.

We have developed an an approach called *PARIMA* along with a pyramid-based bitrate allocation as a learning model to predict next viewport. We have also designed an end-to-end video streaming platform that provides comprehensive evaluation of our model

Contents

Chapter 1

Introduction

360° videos have been capturing attention in industry [1, 2, 3] as well as academia [4, 5, 6, 7, 8] due to their immersing and fascinating experience.360° videos captures view in every direction at the same time and are recorded using omnidirectional cameras. During playback on normal flat display the viewer has control of the viewing direction like a panorama. It can also be played on a displays or projectors arranged in a sphere or some part of a sphere. Demand of 360° cameras has escalated with the rise of panoramic photography, robotics and virtual reality.

Popular streaming platforms like Facebook[2], Youtube[1] and Vimeo[9] have also introduced 360° streaming as a part of their website and application. These platforms use tiling-based or cubemap-based methods for transferring the video frames but they send the entire frame to the client side. This requires significant amount of bandwidth. To tackle this issue, currently these platforms use extensive processing of the 360° videos using optimised task-loading and specialised hardware. Despite the efforts, the processing tasks are of long duration and are CPU and memory intensive. Bandwidth as well as computation and memory requirements can be reduced if there can be some one-time pre-processing of the video at the server side, followed by minimal processing at the client side during streaming time.

With increasing popularity, 360° videos will attract even more user in the coming days. However, one of the major disadvantages of streaming such videos is the need for considerable bandwidth to provide high quality experience to users. Each frame of a 360° video covers an entire sphere, yet, only a small portion of the video(viewport) entertains the user. Roughly 75% of the data transfer for streaming the entire video gets wasted. A major factor that contributes to the problem is that most of the 360° streaming occurs over mobile devices, which use WiFi to get the video contents, making it even more difficult to efficiently acquire the video. Buffer for storing the video at client as well as server side

is much larger in case of 360° videos. Other computational and streaming overheads also increases.

As a consequence, researchers in the field of 360° videos are exploring effective ways to reduce bandwidth consumption yet provide enriching experience to the users. The main focus of research is to predict the viewport beforehand and stream that part of the video with higher quality while reducing the quality of the other parts of the video [6, 8, 10]. However, abrupt change in the quality of video is undesirable. Hence, QoE of the user is also an important factor that is considered while streaming the videos. Researches have discussed prediction of future bandwidth based on the network conditions to adjust the quality of the streaming video accordingly. However, most of the existing documents in literature does not consider the video content while predicting the viewport, rather they only use previous viewport information, that is the head movement direction of the users. Most of the online available 360° video datasets used in research papers use videos that are mostly static i.e which have only one prime object and where the viewport doesn't change by a large amount. Some examples for this are videos in a roller-coaster, concert etc. Generalization of such algorithms to videos containing multiple moving prime object is not feasible.

In this project, we try to incorporate the use of video contents along with the previous viewport information while predicting the next viewport of the user. Such a method will generalize the concept of 360° streaming for videos having dynamic objects. An important observation lies concealed in our argument of using video contents as well as user head movements to predict the next viewport. *We claim that the viewport of the user majorly depends on the trajectories of prime objects in the video.* Thus, we build a model that captures head movements of users to find their viewports and uses object trajectories to learn the FOI(field of interest) of the user in an online fashion.

1.1 Motivation

The project is motivated by the need of having efficient algorithms or methods to stream 360° videos. The following points motivate our project:

1. Sending entire frames over the network in case of 360° videos leads to high bandwidth consumption. This is because each frame considers the entire 360° view and hence, frame size is large.

2. Researches in case of viewport prediction have been based on specific 360° videos which do not have multiple moving objects and these prediction methods don't

consider video content at all. This limits the model of viewport prediction to specific types of videos.

3. Our claim is that viewport of the user depends majorly on the trajectories of the prime objects in the video. The users can watch any object of the video randomly, hence we need an online learning algorithm that can adjust to the user's preferences based on some video preprocessing to find those trajectories.

1.2 Contribution

The contribution of this project is that we developed a 360° video streaming platform that can predict the user's interests dynamically to transfer the video in an adaptive manner so as to reduce the bandwidth consumption. The following are the features of the streaming platform:

1. The video is temporally divided into chunks, and each chunk is further divided spatially into tiles. From the user viewport data, our model PARIMA predicts the viewport of the next chunk(s).

2. The tiles of the chunk are then allocated bitrate adaptively based on the prediction made by the model.

3. During training of model, a one-time processing to detect objects and produce object trajectories from each video is performed.

4. We then evaluate our model extensively against several baselines to support the claim

This report in organised as follows:
In Chapter 2, we discuss the related work based on adaptive 360° video streaming. In Chapter 3, we discuss the background of how 360° video streaming works and how it is different from the general video streaming. Along with this, we discuss other concepts used in the system design. In Chapter 4, we discuss the System Overview, with the detailed design of the system, including video preprocessing, learning model, bitrate allocation and video streaming details. In Chapter 5, we discuss the evaluation testbed, consisting of the different datasets, baselines and metrics used for evaluation. In Chapter 6, we discuss the evaluation results of the experiment and comparison with the baselines. Finally, in Chapter 7, we discuss the conclusions and the future work in the field of 360° streaming.

Chapter 2

Related Works

In traditional HTTP-based adaptive streaming, a video is partitioned into temporal segments and each segment is streamed with the desired quality optimizing the bandwidth as well as QoE of the user. The streaming quality only depends on the network congestion, bandwidth and the QoE. In viewport-adaptive streaming [11, 12, 13], each temporal segment is further divided spatially into tiles. Viewport prediction algorithms require the model to learn the movements of a user and decide which tiles cover the user's future viewport. DASH (Dynamic Adaptive Streaming over HTTP) [14] is a streaming standard which adaptively streams video based on the link bandwidth between server and client.

For adaptive bitrate streaming and to enhance user's quality of experience (QoE), H Mao et.al.[15] has proposed a reinforcement learning-based technique that learns the adaptive bitrate (ABR) algorithms. It learns a control policy for bitrate adaptation purely through experience. Thus, it focuses mainly on network congestion and bandwidth to optimize QoE without considering the viewport. Almquist [16] in his work has mentioned an important trade-off for the amount of video to be stored in buffer. Due to possible future bandwidth fluctuations, the client needs to store a considerable amount of video for smooth experience. However, viewport prediction is done close to playback deadlines, thus not allowing large amount of video to be kept in the buffer. This work has proposed a data-driven characterization of the trade-offs associated with different categories of 360 videos, providing both qualitative and quantitative insights regarding how best to address these trade-offs.

However, an important aspect in adaptive video streaming lies in the prediction of viewports. Several studies have been made to predict viewport effectively. Fan et.al.[5] in his studies developed a fixation prediction network (LSTM) that learns the sensor related features along with image saliency map to predict viewer fixation in the future. On similar grounds, saliency map based viewport prediction has been in the literature

[7, 17]. PanoSalNet [7] learns the saliency map from user viewport data to enhance the prediction accuracy and hence QoE.

Qian et.al.[18] introduces a system named as *Flare* that streams video by predicting head movements of the user. By applying a variety of regression algorithms, they predict the specific portions of a video frame which needs to be fetched. On similar lines, Xie et.al. [19] in his work has approached the issue of adaptive streaming by clustering users and then predicting the next viewport based on the history. Additionally, he presents a QoE-driven rate allocation to minimize the expected streaming distortion under bandwidth constraint. They assign bitrate to each tile by solving a constraint optimization problem. Recent studies like *DRL360* [20] uses deep reinforcement learning based framework to optimize multiple QoE objectives across a broad set of dynamic features, while *Mosaic* [6] makes the use of a CNN+LSTM network to find a tile probability map using saliency map and user head movement logs as inputs.

A recent study, Pano[8] proposes a variable sized tiling scheme to strike a balance between perceived quality and video encoding efficiency, alongwith robust PSNR based quality adaptation algorithm. Variable sized tiling scheme has also been explored in [21].

However, most of the above mentioned works have used user head movements to predict the next viewport. However, none of the previous works have explored the video contents as well as the head movement of the user to stream the video. Saliency maps has been used as a measure of video content [7, 6, 5], which however depends on the most frequent viewed portions and hence user fixation points. It does not decouple the video contents completely. In this report, we propose an algorithm that would decouple the video contents from user fixation points and predict the user viewports while streaming the video in an online manner.

Chapter 3

Background

In this chapter, we would like to discuss the basics of 360° video streaming, how it works and how it is different from the normal video streaming. Along with this, we would discuss other concepts necessary for the system description.

3.1 360° Video Creation

Figure 3.1: *Equirectangular Projection of a 360° Video Frame*

360° video is typically recorded using either a special setup of multiple cameras, or using a dedicated omnidirectional camera that contains multiple camera lenses embedded into the device. Separate video footage are then *stitched* to form one spherical video piece such that the color and contrast of each shot is consistent with the others. The stitching algorithms might require careful calibration and time-consuming feature matching to ensure proper overlapping. The large size of frames due to wide view and the processing of

video is the reason behind the large size of 360°videos

The typical way to store a 360° video is by converting it to an equirectangular projection. Equirectangular projection exhibits stretching of objects near the top and the bottom while object shape is preserved near the middle of the frame. Figure 3.1 shows a typical equirectangular projection of a 360° video. Other common projections are fisheye projection and stereographic projection. We have used equirectangular projection for all purposes in the project.

3.2 Streaming of 360° Video

Any video streaming platform is typically a client-server system. The server stores the entire video and the client streams the video. At the back end of client side, it requests the server for chunks of video frames and the server simply processes the requests and returns the frames. The entire server-client communication may occur through Ethernet, WiFi or cellular networks.

A general video is simply processed and streamed in the above manner. Adaptive video streaming techniques for regular videos focus on optimising bandwidth consumption based on the network. Videos are temporally partitioned into segments and each segment is streamed with the desired quality. The quality of stream mainly depends on the network congestion and bandwidth optimization.

A general video is simply processed and streamed in the above manner. Adaptive video streaming techniques for normal videos focus on optimising the bandwidth consumption based on the network.

On the other hand, a 360° video requires some additional features. For 360° videos, each temporal segment is further divided into spatial tiles. The tile that the user looks at forms the viewport. If each temporal segment is streamed at the same quality, a significant amount of bandwidth is consumed, since any user looks only at some of the spatial tiles. Thus, apart from the quality of each temporal segment, each spatial tile also needs to be streamed at different quality.

360° videos are usually viewed using personal computers, mobile devices or head-mounted displays. In 360° videos, the user can dynamically change his/her viewport for immersing experience. Mobile devices use gyroscope to decide the viewport based on the orientation of the device, while users can click and drag the videos in personal computers. Thus, whenever new frames are sent by the server to the client, they are streamed to the client based on these orientations. Also, since the 360° videos have a wider field of view, the size of each frame is larger in general. Hence, streaming a 360° video typically

requires larger bandwidth and short response time.

The need for optimizations over 360° video streaming are emerging as a result of the large number of novel applications that they have [22]. These include socialising with friends in a virtual room, virtually walking the potential house buyers through the home, remote medical training for surgeries, etc.

3.3 Image Projections and their Inter-conversion

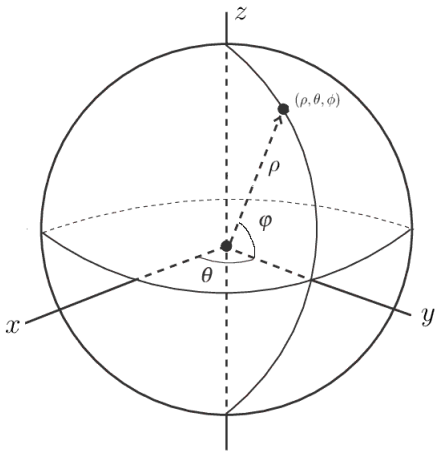


Figure 3.2: *Spherical Coordinates (ρ, θ, ϕ), where ρ is the radial distance, θ is the polar distance, ϕ is the azimuthal angle. θ and ϕ shown here in this figure is considered in positive direction*

This section discusses different image projections for 360° videos and how one projection can be converted to the other. The viewport of a user can be represented as Cartesian coordinates in spherical space. However, for image processing, we need to represent the 360° video in a 2-D plane. The reason for this is that due to the distortion in the shapes of the objects, 360° videos cannot be processed in a 3-D plane. In a typical 360° frame, the objects near the centre vertically are less distorted and the objects near the ends are more distorted.

The most popular way of representing a 360° image in a 2-D plane is by using its 'Equirectangular Projection'. The projection maps meridians to vertical straight lines of constant spacing, and circles of latitude to horizontal straight lines of constant spacing. Cartesian coordinates in a sphere can be easily converted to equirectangular coordinates. Coordinates in equirectangular projection is simply the longitudes and latitudes of the sphere which is a trivial modification of the polar and azimuthal angles of a point in a sphere.

The convention for the polar and the azimuthal angles that we follow throughout the report is shown in Figure 3.2

Let (x, y, z) be the Cartesian coordinates of a point in sphere. We can get its repre-

sentation in spherical coordinates system (ρ, θ, ϕ) using the following transformation:

$$\rho = \sqrt{x^2 + y^2 + z^2}$$
$$\theta = \arctan \frac{y}{x}$$
$$\phi = \arcsin \frac{z}{\rho}$$

Thus, the longitude(L_n) and latitude(L_t) can easily be computed as:

$$L_n = \frac{180°\theta}{\pi}$$
$$L_t = \frac{180°\phi}{\pi}$$

The above transformation from (x, y, z) to (L_n, L_t) gives us a representation of a 3-D image in 2-D plane. However, it is evident that objects in the equirectangular projection will be distorted, especially near the poles. Hence, any object detection algorithm will not be able to detect all the objects accurately. Thus, we further introduce a transformation that maps an image in 'Equirectangular Projection' to 'Cube map Projection'.

In computer graphics, Cube map projection is a method of environment mapping that uses the six faces of a cube as the map shape. The environment is projected onto the sides of a cube and stored as six square textures, or unfolded into six regions of a single texture. Transformation from spherical space to cube map projection can be easily understood using the spherical coordinates (ρ, θ, ϕ) instead of Cartesian coordinates (x, y, z).

We have $\theta \in [-\pi, \pi]$ and $\phi \in [-\pi/2, \pi/2]$. Thus the front face of the cube can only capture the pixels of the image having $\theta \in [-\pi/4, \pi/4]$. The central projection of a point $(\rho\cos\phi\cos\theta, \rho\cos\phi\sin\theta, \rho\sin\phi)$ on the sphere will be $(t\cos\phi\cos\theta, t\cos\phi\sin\theta, t\sin\phi)$ which hits the plane $x = \rho$ when $t = \frac{\rho}{\cos\phi\cos\theta}$.

Hence the projected point is $(\rho, \rho\tan\theta, \rho\tan\phi\sec\theta)$.

If $|\tan\phi\sec\theta| < 1$, then the point will be projected on the front face of the cube. Otherwise, it will be projected either on the top or the bottom face of the cube, which would then require further computations of a different projection which would hit the plane with $z = \rho$ or $z = -\rho$.

However, it is easy to notice that whenever $\phi > \pi/4$ or $\phi < -\pi/4$, the point will always be projected to the top or bottom face of the cube. Similar will be the arguments for projecting a point on the other faces of the cube.

Thus, for any point $(\rho\cos\phi\cos\theta, \rho\cos\phi\sin\theta, \rho\sin\phi)$, its projected point on any of the face of the cube will be:

Face	Coordinates	Conditions
Front	$(\rho, \rho\tan\theta, \rho\tan\phi\sec\theta)$	$\forall\theta \in [-\pi/4, \pi/4], \phi \in [-\pi/4, \pi/4]$
Right	$(\rho\cot\theta, \rho, \rho\tan\phi\theta)$	$\forall\theta \in [\pi/4, 3\pi/4], \phi \in [-\pi/4, \pi/4]$
Back	$(\text{-}\rho, -\rho\tan\theta, -\rho\tan\phi\sec\theta)$	$\forall\theta \in [3\pi/4, \pi] \cup [-\pi, -3\pi/4], \phi \in [-\pi/4, \pi/4]$
Left	$(\text{-}\rho\cot\theta, -\rho, -\rho\tan\phi\theta)$	$\forall\theta \in [-3\pi/4, -\pi/4], \phi \in [-\pi/4, \pi/4]$
Top	$(\rho\cot\phi\cos\theta, \rho\cot\phi\sin\theta, \rho)$	$\forall\phi > \pi/4$
Bottom	$(\text{-}\rho\cot\phi\cos\theta, -\rho\cot\phi\sin\theta, -\rho)$	$\forall\phi < -\pi/4$

Table 3.1: *Equirectangular to Cube map projection*

The above transformations are also invertible and can be used to convert back Cube Map Projection to Equirectangular Projection and Equirectangular to Spherical Projection. These would be primarily used in the system for video preprocessing tasks.

We state a minor result of solid angle calculation in this paragraph which will be used later. In our framework, if an object moves from point (x_1, y_1, z_1) to (x_2, y_2, z_2) on the surface of a sphere, then assuming that the distance between the two points D is small, we calculate the solid angle using the formula

$$\Omega = \frac{\pi D^2}{4\rho^2} \tag{3.1}$$

3.4 Bitrate

In simple terms, bitrate of a video being streamed is just the number of bits transferred per second [23, 24]. It determines the quality and size of the video transferred. A higher bitrate would mean a larger number of bits that can be transferred for a specific length of video, thus impacting its quality in a positive way. The size of video transferred is simply the product of bitrate and the video duration. Having a higher bitrate would accommodate a higher quality image. However, the quality of the video is also limited by the pixel resolution of the actual video, and after a certain point, it cannot be improved further since the pixel resolution will be limited.

To give an illustration, refer to Figure 3.3. Suppose we are streaming three different videos in three different cases, with the specifications mentioned in the figure. Suppose the video is to be streamed at 8 Mbps.

For the first video having 30 frames per second, streaming will take place such that 8 Mb of data is assigned to 30 frames, that is, each frame gets a proportion of the 8 Mb of data. Among this proportion, each part of the frame gets a proportion of the

bitrate allotted to that frame, such that the entire 1 second chunk gets coded into 8 Mb. In contrary, the second video has a higher number of frames per second with the same frame dimensions. This would result in getting a lesser amount of bitrate for each frame within a second, which eould decrease its video quality in comparison to Video 1. However, in the third video, the number of frames per second is the 30 with frame dimensions being larger than both the videos. Therefore, in this case too, each part of the frame will get a lower proportion of the bitrate and hence exhibit poorer quality.

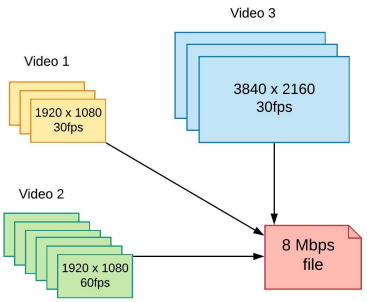


Figure 3.3: *Video Streaming Bitrate Demonstration*

For 360° videos, the frame size is typically large. Hence, for the same bitrate for streaming, a normal video will be streamed in higher quality than a 360° video. In other words, to maintain the same quality for streaming, a 360° video would require higher bitrate than a normal video. Hence, the bandwidth consumption is higher for 360° videos.

360° videos are typically divided spatially into tiles. Through viewport adaptive video streaming, we intend to provide higher bitrate to the tiles that contends to be the viewport, so that the user viewport is at a higher quality than the non-viewed regions of the frame.

Chapter 4

System Description

Streaming 360° videos involves a collection of tasks that needs to be performed in order to provide the best user experience. Our methodology involves the use of video contents in order to accurately predict the following viewport. Video contents is specified primarily by the objects present in the video and their trajectory. Thus, the task of finding the contents involve a detailed analysis of the trajectories of each objects present in the video. Current viewport of the user is detected by capturing the movement of the device playing the video. Based on the set of viewports observed recently and the video contents, the model predicts the next viewport. Based on the predictions, it finds the bitrate to be allocated to each tile and then requests the corresponding tiles at the given bitrates from the server. Before making the next set of predictions, the client will also update the model to fit to recent user preferences.

4.1 Video Preprocessing

Since we claim that the user viewport primarily depends on the trajectory of prime objects present in the video, it is important to extract details about the important contents of the video which can be used to predict the viewport. The steps taken for the one-time preprocessing of the video to extract the object trajectories is shown in Figure 4.1. Our methodology for video preprocessing to extract meta-data from the video consists of the following parts:

1. Conversion of equirectangular frames to cube map projection according to Section 3.3. This is to obtain a less distorted version of the frame contents.

2. Stitching cube map projections to get a continuous two-dimensional view of the 360° frame.

3. Object detection in stitched cube map frames to obtain a list of objects for each frame

4. Re-projection of the coordinates of bounding boxes of the detected objects back to equirectangular form

5. Tracking object trajectories using centroid tracking in spherical projection space

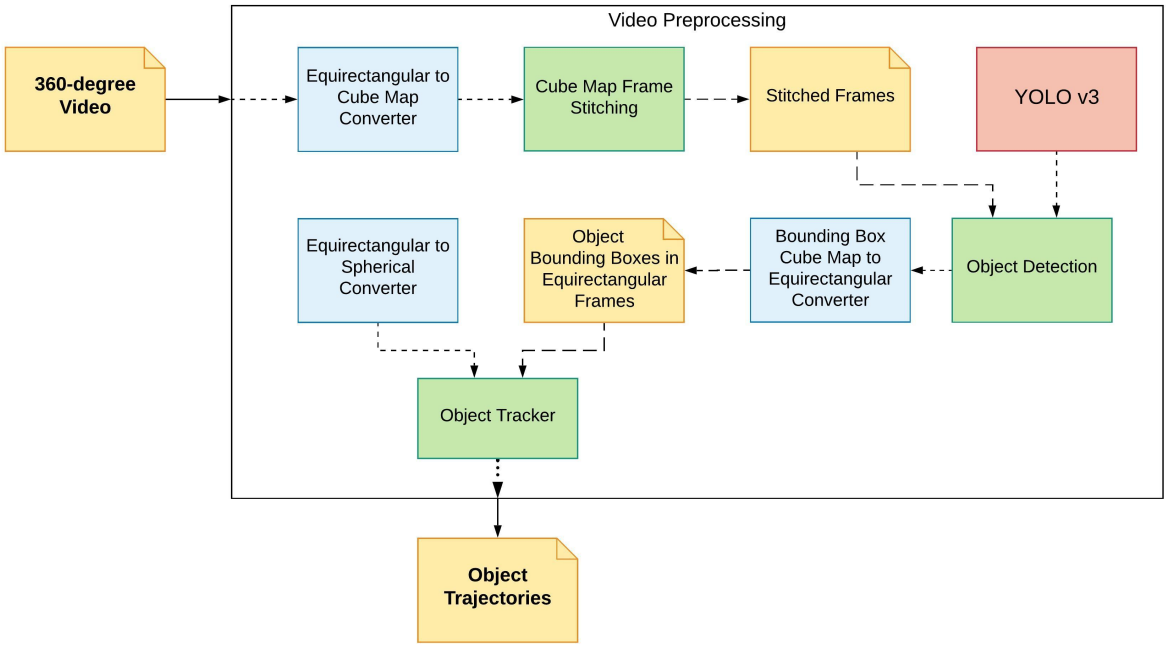


Figure 4.1: *Schematic diagram of the preprocessing steps used to extract object trajectories from the video.*

This section discusses the above steps in detail. The input to this module is a 360° video stored in equirectangular projection form and the final output will be a list of prime objects and their trajectories, which are a representation of the video contents.

4.1.1 Equirectangular to Cube Map Conversion

As discussed in section 3.3, an equirectangular frame obtained from the spherical space for a 360° video, is distorted. The distortion is less towards the centre and more towards the poles. For example, in Figure 3.1, the bag present at the bottom is split across the width of the frame because of the nature of the projection. Hence, in equirectangular projection, detection of objects is an issue because of the non-uniform distortion as no model can

be trained to detect randomly distorted objects. Hence, we convert the equirectangular frames to cube map projection [25]. An example for cube map projection is shown in Figure 4.2(a)-(f).

4.1.2 Frame Stitching and Object Detection

(a) *Left* ***(b)*** *Front* ***(c)*** *Right*

(d) *Back* ***(e)*** *Top* ***(f)*** *Bottom* ***(g)*** *Stitched image*

Figure 4.2: *Figure shows the different faces of a cube map projection and their stitched image. In the stitched image, each adjacent face of the cube is placed in such a way that they share a common boundary. Top and Bottom face of the cube is rotated the appropriate amount so that they form a continuous image*

After the conversion of the Equirectangular projection to Cubemap projection of the frames, we need to perform object detection. However, one noticeable disadvantage of object detection on each of the cube faces for a frame is evident from the definition of the projection. Since each pixel of the equirectangular image is allocated a unique face of the cube, a single object might split and get mapped to different faces depending on its location on the sphere. Hence, any object detection algorithm will either not be able to detect the object or will detect as two objects for the adjacent faces. Cubemap projection does not handle this event of an object being mapped to two adjacent faces of the cube.

To overcome this issue of single object mapped to multiple faces of the cube, we stitch the different faces of the cube to form a single image (Figure 4.2). Stitching ensures that any object mapped to adjacent faces of the cube gets treated as an entire entity. This ensures that object detection and tracking is continuous and doesn't create different objects for the same entity. Apart from this, the object tracking algorithm will become more robust and will ensure that even if an object is missed in few frames and is detected back in the next frame, it will be able to map the same and interpolate the object position for the frames where the object was missed to maintain continuity.

We have used YOLOv3 [26] algorithm on the stitched image of each frame to detect the objects and get their bounding box coordinates. YOLOv3 is a pre-trained neural network based object detection model that predicts an objectness score for each bounding box using logistic regression. The objectness score is 1 if the bounding box prior overlaps a ground truth object by more than any other bounding box prior.

After the bounding box coordinates for each object is obtained for a frame (stitched image), it is then translated back to its equirectangular projection. This is because an object might wrap around the edges of a frame, which will prohibit object tracking to ensure correct results.

We implement the tracking algorithm in spherical space to take care of the wrapping-around issue. Hence, with an inverse mapping of the equations in Table 3.1, we get the desired output.

4.1.3 Object Tracking

Once we have detected objects in different frames and obtained their bounding boxes in equirectangular projection, we then need to track object trajectories by assigning them IDs such that objects in one frame can be mapped to objects in the next frame based upon some algorithm. We claim the following points in regard to object tracking:

- There exists no deterministic algorithm that exactly predicts the trajectory of an object in a video. This is because there are potentially infinite possibilities of different objects in different frames that can occur in a 360° video. Hence, detecting the exact trajectory of each object is not possible. An algorithm which approximately maps the objects to some IDs will work because of the online nature of the learning algorithm.

- Due to the fact that no projection in 360° frames produces a completely undistorted image, object detection algorithms may not be able to exactly identify the objects. It might also happen that the objects have distorted bounding boxes. Due to these limitations of object detection algorithms, we should have a robust approximation that can interpolate object positions even if the object goes missing in between for a few frames and assign them the same id. Thus, object tracking algorithm should complement the weaknesses of the object detection algorithm, typically the object missing in between for some frames.

- General object tracking uses 2-Dimensional space. That is, if an object disappears at one end and reached the other, they are not considered as the same objects. No

general method for tracking in 360° videos exists.

We have used an advanced 360° version of centroid-based object tracking algorithm [27] in spherical space to get the object trajectories. The reason for using spherical projection is that even if an object wraps around over equirectangular frames, it will have a small solid angle in spherical projection. Hence, instead of using Euclidean distance, we would prefer to use spherical distance. The algorithm is fairly robust if bounding boxes of some object go missing for some frames in between, up to a limit of 30 frames. If within the 30 frames limit, the object reappears, then it is assigned the same ID as before and the object positions are interpolated for the missing frames. The following is a high-level description of the algorithm:

1. Get the list of coordinates of objects for each equirectangular frame and compute their centroid.

2. For the initial frame, allocate each object a unique ID. These can be called as the currently active objects.

3. For the subsequent frames, perform the following steps to associate objects of the current frame to the previous frames:

   (a) Project each of the centroids of the new and currently active object IDs to spherical projection as mentioned in Section 3.3.

   (b) Find the solid angle (Equation 3.1) between all pairs of new centroids and existing centroids

   (c) It is assumed that within two consecutive frames, the object is expected to move minimally. Hence, for each newly detected objects, find the active object which is nearest to it and for which, it is nearest to the active object. Assign the new object the same ID as the corresponding active object.

   (d) In the above case, some new objects may not be assigned any active object and some active object may not be assigned to any new object. This leads to two new methods:

      i. **Register New Object:** If a newly detected object is not assigned any existing active object, means that it has appeared for the first time in the video. In this case, we assign the object a new ID and register it as an active object. We would note the start frame of the object's appearance.

      ii. **De-register Old Objects**: If an active object is not assigned to any newly detected object in the current frame, it means that the object has

disappeared/not detected. In this case, we maintain a count of the number of successive frames for which the object is undetected. If the count crosses 30, we declare the object to be disappeared or removed and the object is no longer active. We would note the last frame of the object's appearance. However, if it reappears within 30 frames, it will be assigned the old object ID and the count is reset to 0.

4. Finally, we make a pass through all the frames, checking the object IDs and interpolating the coordinates for the active objects which were undetected in between.

At the end of this section, for each frame, we have a set of objects, each indexed by an ID, such that over multiple frames, the same object would be assigned the same ID.

4.2 Viewport Prediction

Predicting the upcoming viewport from the video meta-data and the previous viewports is a challenging task, especially when we want to model a dynamic system based on the content of the videos. Upcoming viewport not only depends on the trajectories of the prime objects, but also remains in the vicinity of the previous viewport. As a consequence, the learning task must be online, learn viewport based on object trajectories and previous viewports, and the weights must be updated in an incremental manner in order to be fast and accurate and can quickly adapt to the changes in actual viewport. This instils the idea of user preference in the model which is captured by the information about the previous viewport.

An important and obvious requirement of the streaming model is for it to be able to predict multiple frames in the future at the same time and stream them in the form of chunks rather than a single frame at a time, to maintain the quality of experience for the user. For this, the video is segmented temporally into chunks of duration t seconds, which are further segmented spatially into $m \times n$ tiles. There is a trade-off between the prediction accuracy and network bandwidth consumption [16] while deciding the chunk size. A larger chunk size facilitates total network consumption while compromising prediction accuracy, since user viewport tend to vary within the chunk duration while these changes would get reflected only after an entire chunk is completed. Studies by Qian et.al.[18, 28], showed that 1 second chunk duration is optimally good for streaming in viewport adaptive case which we have used. We also experimented our model with different chunk sizes, the results and analysis of which is present in Chapter 6.

Before building our model, we pre-compute the trajectories of all the objects detected in the video using the object tracking algorithm. We have primarily used Passive Aggressive Regressor[29] and ARIMA model to predict viewport individually. However, to enhance the prediction accuracy, we have combined the two models effectively to capture their pros. This section discusses the details of the two models and their augmented combination.

The viewport prediction algorithm will take the viewports of a set of previous frames and the object trajectories of the frames corresponding to the upcoming one second as the inputs. Based on these inputs, it would output a set of predictions for the next chunk in the form of (F_i, α_i, β_i) where F_i is the frame number, α_i and β_i are the x and y coordinates of the viewport, that can be converted into corresponding tile numbers. Based upon the actual viewports observed later and the predictions, the model updates itself to adapt to the user preferences. .

4.2.1 Passive-Aggressive Regression Model

Passive Aggressive Algorithms are well known online learning algorithms. Passive-Aggressive Regression [29] as a general regression algorithm, computes the mapping $f : \mathbb{R}^n \rightarrow \mathbb{R}$, $f(\mathbf{x}; \theta) = \theta^T \mathbf{x}$ where, $\theta, \mathbf{x} \in \mathbb{R}^n$. The algorithm uses the Hinge Loss Function, given by:

$$L(\theta, \epsilon) = max(0, |y - f(x_t; \theta)| - \epsilon)$$

The Hinge Loss Function is called ϵ-insensitive. When the predicted viewport and the actual viewport differ by a value less than ϵ, then the algorithm is said to act as *passive*. Otherwise, the algorithm is said to act as *aggressive*, because it looks for a set of weights that ensure that the prediction is as close as the previous one. Hence, the parameter ϵ determines a tolerance for prediction errors. The weight update rule for PA Regression is:

$$\theta^{t+1} = \theta^t + \alpha \frac{max(0, |y_t - \theta^T x_t| - \epsilon)}{||x_t||^2 + \frac{1}{2C}} sign(y_t - \theta^T x_t) x_t$$

C is a slack variable that allows to have soft margin and α is the learning rate. Higher C values yields stronger aggressiveness, while lower values allow better adaptation.

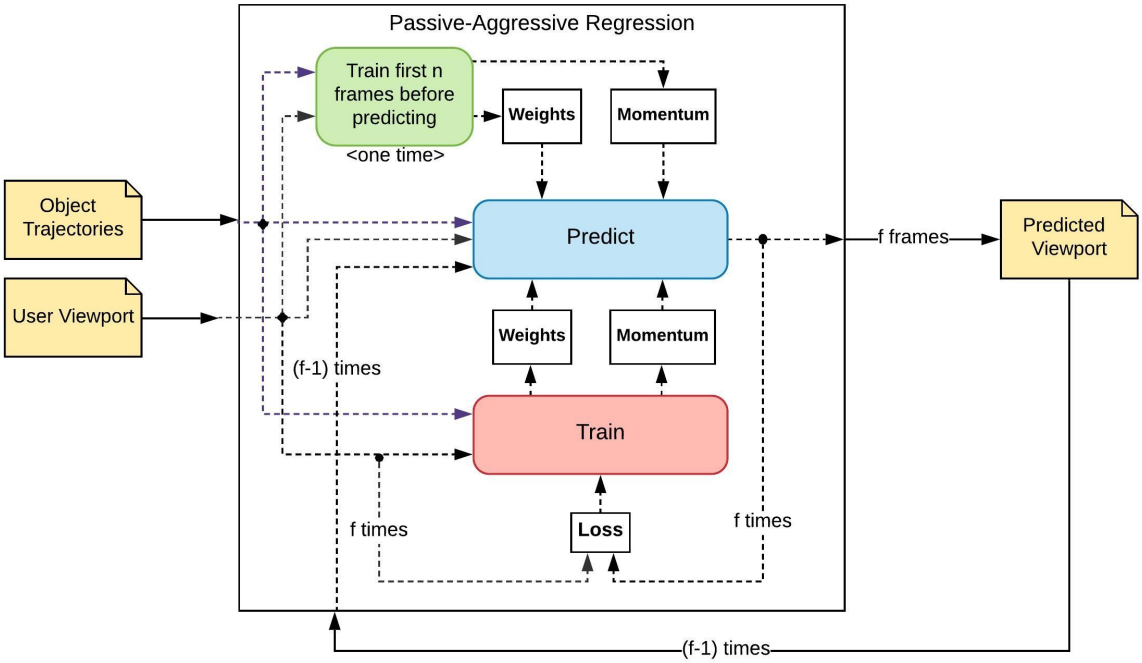


Figure 4.3: *Schematic Diagram of the Passive-Aggressive model used. It takes as inputs the object trajectories and the user viewports and outputs the predicted viewport.*

The central Passive-Aggressive Regression model (Figure 4.3) takes as input the coordinates of the different objects in the frame as well as the viewport of the previous frame and predicts viewport of the next set of frames. An algorithmic definition of the steps our formulated in Algorithm 1. In order to prevent the initial zero prediction of the model, we train the model for the first $5 * fps$ frames at the beginning. Thus, for these set of frames, instead of making predictions, we send all the tiles at uniform quality. Based on the viewports observed in those frames and object trajectories, we train the model to adapt to the user preferences to make the predictions for the following chunk of frames.

We use the set of predicted frames to calculate the bitrate to be allocated to the chunk and query to the server. Once this predicted frames gets rendered, we will use the actual viewports from the user to train the regression model again and repeat the process for predicting. A second order momentum ν is used to supplement the accuracy of the predictions. At t^{th} iteration,

$$\nu_t = \rho\nu_{t-1} + (1-\rho)\Big\{\frac{max(0, |y_t - \theta^T x_t| - \epsilon)}{||x_t||^2 + \frac{1}{2C}} sign(y_t - \theta^T x_t) x_t\Big\}^2$$

The advantages of Passive-Aggressive regression over other forms of naive regression methods are manifold. Firstly, Passive-Aggressive models are explainable, unlike other

Algorithm 1 Passive-Aggressive Regression based Viewport Prediction

Input: Object Trajectories O_f $\forall f$ frames, Streaming Viewport of the user V_f, PA model M

Output: Predicted Viewport for all frames

procedure GET_PA_VIEWPORT(O, V, M)
  Initial Training of $5fps$ frames on model M.
  List of Predicted Viewports PV
  for each chunk c **do**
    $F^c \leftarrow$ frames in chunk c
    Predicted Viewport for first frame ${}^cF_1$ in chunk c $(PV_{F_1^c}) \leftarrow M(O_{F_1^c}, V_{F_1^c})$
    Append $PV_{F_1^c}$ in PV
    for frame $F_f^c \in [F_2^c, F_c^c]$ **do**
      $PV_{F_f^c} = M(O_{F_f^c}, PV_{F_{f-1}^c})$
      Append $PV_{F_f^c}$ in PV
    end for
    Train M on inputs $O_{F_1^c} \rightarrow O_{F_c^c}$ and $V_{F_1^c} \rightarrow V_{F_c^c}$
  end for
  return: PV
end procedure

machine learning models like neural networks, which might not be generally explainable in our case. The coefficients of the parameters in the PA-Regression Model essentially represent the user preferences at that moment of time. If the coefficient corresponding to a certain object is higher at some time, it would mean that the user is more inclined to see that object.

Secondly, Passive-Aggressive Regression model suits better as an online learning algorithm since at every model update step, it tries to update the model weights in such a way that they take the predicted value as close to the actual value as possible. Hence, it adapts better. Also, the prediction and model update time is much faster than other models. Experimental verification in Section 6.1 justifies the choice of PA Regression over other regression methods.

However, Passive-Aggressive Regression alone can sometimes lead to larger prediction errors while predicting multiple frames. This is because while predicting multiple frames, the input given to the model is the predicted viewport from the previous frame. Hence, even with slight deviations in predicted viewport at every step, the error can become very large eventually, since the error cascades with every prediction. Hence, it might

overshoot in some cases.

4.2.2 Time Series (ARIMA) Model

Time series models are often used in predicting head movements of users. Hence, it can also be used to predict the next viewport of a user while watching a 360° video, because next viewport is essentially a temporal function of user head movement. Compounded with the applicability, as mentioned already, in order to mitigate the issue of high errors of Passive-Aggressive counterpart, a time series model, specifically ARIMA model is used to predict user head movements of the subsequent frames.

ARIMA model of the order (p, d, q) can be written as

$$\Phi_p(B)(1-B)^d X_t = \Theta_q(B) Z_t$$

Here, p = order of AR process, q = order of MA process, and d = degree of differencing

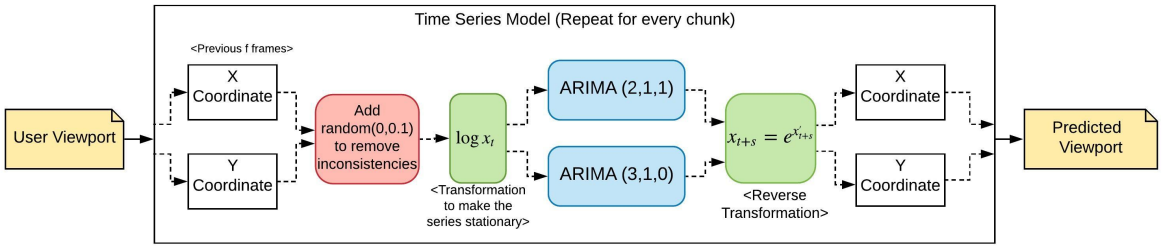


Figure 4.4: *Schematic Diagram of the ARIMA model used. It takes user viewports as inputs and outputs the predicted viewport.*

The model (Figure 4.4) that we have considered here takes as input the x and y coordinates (horizontal and vertical components respectively) of the previous viewport data of the user [Here $x \in (0, \text{width}), y \in (0, \text{height})$]. We add *random(0,0.1)* to the viewport coordinates in order to remove inconsistencies from the data that prevents the entire viewport data of a chunk from having identical values, thus catering to a positive semi-definite auto-covariance matrix. The algorithmic steps are mentioned in Algorithm 2.

Augmented Dickey-Fuller test [30] on the raw viewport data proved that the time series is not stationary. However, ARIMA models require stationarity in the time series. Thus we project the series into logarithmic domain and furthermore, the series becomes stationary, once a one-degree differencing is applied on $\log x_t$ to get $log(x_t/x_{t-1})$. This differencing is achieved by the ARIMA model.

Algorithm 2 ARIMA based Viewport Prediction

Input: Streaming Viewport of the user V_f $\forall f$ frames, ARIMA model M_x, M_y

Output: Predicted Viewport for all frames

procedure GET_ARIMA_VIEWPORT(V, M)
    List of Predicted Viewports PV
    for each chunk c **do**
        $F^{c-1} \leftarrow$ frames in chunk $c-1$
        $(X_{F_1^{c-1}}, Y_{F_1^{c-1}}) \leftarrow$ horizontal and vertical components of $V_{F_1^{c-1}}$
        Make series $(X_{F_1^{c-1}}, Y_{F_1^{c-1}})$ stationary
        Train M_x on inputs $X_{F_1^{c-1}} \rightarrow X_{F_{c-1}^{c-1}}$, M_y on inputs $Y_{F_1^{c-1}} \rightarrow Y_{F_{c-1}^{c-1}}$
        $F^c \leftarrow$ frames in chunk c
        for frame $F_f^c \in [F_1^c, F_c^c]$ **do**
            $PV_{F_f^c} \leftarrow (X_{F_f^c}, Y_{F_f^c}) \leftarrow$ Prediction of M_x, M_y
            Append $PV_{F_f^c}$ in PV
        end for
    end for
    return: PV
end procedure

As illustrated, we train separate models for predicting the X and Y components of the viewport. *We rebuild our model for every chunk(shows a greater accuracy because the viewport of an user over the length of video is quite random and depends on the whim of the user, which disrupts the seasonal and stationarity of the time series).* The predicted viewport is then reprojected back from the logarithmic domain to the pixel domain by a simple exponentiation. While rendering the frames of the current chunk, we can log the actual viewport data which is then further used for head movement prediction of the next chunk.

Advantages of ARIMA model over Passive-Aggressive counterpart includes the use of previous chunks while predicting future viewport. This helps in maintaining the locality information which remains an integral part in head movement prediction. Since ARIMA learns on previous data and it does not incorporate the aggressive attribute of Passive-Aggressive regression, the predictions are smoother with significant reduction in error.

However, ARIMA model does not leverage the information of the video content, that is the object trajectory while predicting which was being captured in Passive-Aggressive counterpart. Our claim stands that the user head movement depends on the video content as well. Hence, to aggregate the positives of both the models, we design an augmented

model that would alleviate the prediction error as well as strengthen the quality of experience (QoE).

4.2.3 PARIMA: Augmented PA-ARIMA Model

The model is an combination of Passive Aggressive Regressor and ARIMA time series as shown in Figure 4.5. The model used by our system combines the pros of PA-Regressor and the ARIMA model to give a fairly good model for video streaming.

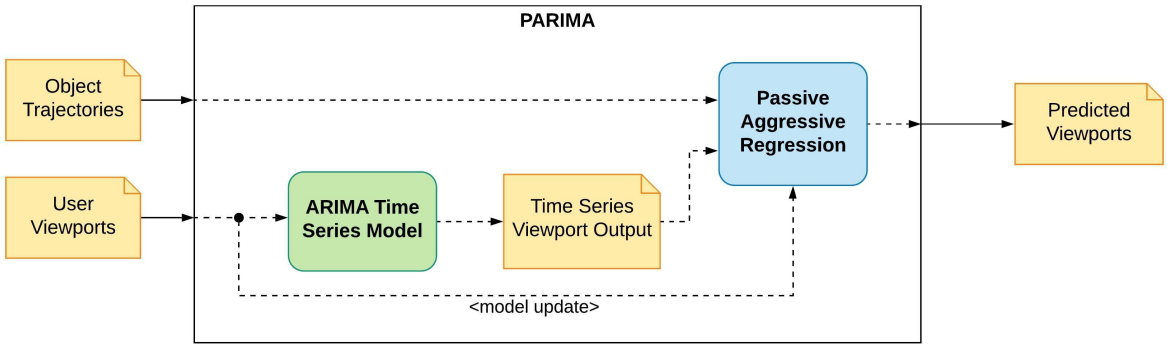


Figure 4.5: *Schematic Diagram of the PARIMA model: Augmented PA-ARIMA model used. It takes user viewports and object trajectories as inputs and outputs the predicted viewport.*

As illustrated in Algorithm 3, we have coupled the PA model and the ARIMA model such that they transfer informations among each other. For a chunk, we first compute the temporary predicted viewport using the ARIMA model as in Algorithm 2. The output viewports of these set of frames are fed as input to the Passive-Aggressive Regressor along with the object coordinates to predict the next set of viewports. Hence, the predicted viewports are essentially a weighted combination of the PA-Regressor output considering only the object trajectories and the time series model, with the weights of the two models being adjusted dynamically based on changing user preferences.

Once the set of predicted frames for a chunk are rendered, the actual viewports of the user are used to retrain the model weights.

It is important to note that the augmented combination of the two models provides a much superior Quality of Experience (QoE) than the two models individually. The results of the experiments are presented in Chapter 7. The model combines the pros of the PA-Regressor and ARIMA Time Series model since it is fast, adapts to the user preferences effectively and doesn't overshoot.

Algorithm 3 PARIMA based Viewport Prediction

Input: Object Trajectories O_f $\forall f$ frames, Streaming Viewport of the user V_f, PA model M^1, ARIMA model M_x^2, M_y^2

Output: Predicted Viewport for all frames

procedure GET_PARIMA_VIEWPORT(V, M)
  Initial Training of $5fps$ frames on model M^1.
  List of Predicted Viewports PV
  for each chunk c **do**
    $F^{c-1} \leftarrow$ frames in chunk $c-1$
    $(X_{F_1^{c-1}}, Y_{F_1^{c-1}}) \leftarrow$ horizontal and vertical components of $V_{F_1^{c-1}}$
    Make series $(X_{F_1^{c-1}}, Y_{F_1^{c-1}})$ stationary
    Train M_x^2 on inputs $X_{F_1^{c-1}} \rightarrow X_{F_{c-1}^{c-1}}$, M_y^2 on inputs $Y_{F_1^{c-1}} \rightarrow Y_{F_{c-1}^{c-1}}$
    $F^c \leftarrow$ frames in chunk c
    for frame $F_f^c \in [F_1^c, F_c^c]$ **do**
      $temp\_PV_{F_f^c} \leftarrow (X_{F_f^c}, Y_{F_f^c}) \leftarrow$ Prediction of M_x^2, M_y^2
      $PF_{F_f^c} \leftarrow M^1(O_{F_f^c}, temp\_PV_{F_f^c})$
      Append $PV_{F_f^c}$ in PV
    end for
    Train M^1 on inputs $O_{F_1^c} \rightarrow O_{F_c^c}$ and $V_{F_1^c} \rightarrow V_{F_c^c}$
  end for
  return: PV
end procedure

4.3 Bitrate Allocation

Bitrate allocation to tiles should be accomplished in a way such that the tiles corresponding to the viewport should get higher bitrate than the off-viewport ones. This would maximize the Quality of Experience (QoE). This way, for the same bitrate, the quality of viewport will be higher.

It is also essential to note that, for a particular chunk, multiple tiles might be predicted as viewports. This is because our viewport prediction is based on each frame. Hence, a mechanism is needed to ensure that all the tiles corresponding to the viewport are given higher bitrate than the rest, and to maintain uniformity, the bitrate should reduce gradually as we move away from the viewport.

To achieve this, we use a *pyramid-based model* of bitrate allocation [13]. The pyramid model gives the highest quality of video to the tile viewed by the user, which gradually

Figure 4.6: *Pyramid Model of Bitrate Allocation. Consider tiles to be numbered as (x,y), where x and y are the horizontal and vertical positions of the tile from the top-left corner. In this figure, An equirectangular frame split in* 8×8 *tile format, which has viewport at tile* $(3, 3)$*. Lower opacity signifies higher proportion of bitrate to be allotted to the tile*

decreases till the tile present opposite to the user viewport as shown in Figure 4.6.

To assign a distribution of bitrates to the tiles (i, j) in each chunk c, we use a weight function that would capture the how much proportion of the total bitrate that should be given to a specific tile (i, j). Whenever a tile (i', j') is a candidate viewport, its weight is increased by a unit and the weights of the other tiles are increased based on a pyramid approach where the farthest tiles weigh the least. This weight function is normalised and then used to assign bitrates to each tile accordingly as formulated in Algorithm 4.

For 360° videos, it must be noted that since the video frame wraps around into a spherical one, the Manhattan distance between two a tile and viewport can be different than in case of a general 2D-video frame. The distance between two tiles in a 360° equirectangular frame is the minimum of the distance seen directly in the frame and the distance obtained by wrapping around through the one of the ends of the frame. For example, in Figure 4.6, consider the distance between tile $(8, 4)$ and $(3, 3)$. If we calculate the Manhattan distance directly, the distance is given by |8 - 4| + |4 - 3| = 6 units. However, when we consider the frame to be wrapped around, the actual distance is |8 - (3 + 8)| + |4 - 3| = 4 units. Hence, we take the minimum Manhattan distance for allocation of bitrates.

Algorithm 4 Assign Bitrates to Chunks

Input: Predicted Tiles for each frame T_f, preferred bitrate B_p

Output: Allocated Bitrates for each chunk B_c

procedure SELECTBITRATES(T_f, B_p)
    Initialize Bitrate B_c
    for each chunk c **do**
        Initialize $weight_{ij}$ for each tile (i, j)
        for each frame f_c in c **do**
            $(i', j') \leftarrow$ tile viewport in frame f_c
            $weight_{i'j'} \leftarrow weight_{i'j'} + 1$
            for all $(i, j) \neq (i', j')$ **do**
                $d_{ij} \leftarrow$ min. manhattan dist. from (i', j')
                if (i, j) within Video Player FoV **then**
                    $weight_{ij} \leftarrow weight_{ij} + 1 - \frac{2 * d_{ij}}{max(d_{ij})}$
                else
                    $weight_{ij} \leftarrow weight_{ij} + 1 - \frac{d_{ij}}{max(d_{ij})}$
                end if
            end for
        end for
        $B_c^{ij} \leftarrow \frac{weight_{ij}}{\sum_{i,j} weight_{ij}} B_p, \forall (i, j)$
    end for
    return: B_c
end procedure

4.4 Video Streaming

This section deals with the actual streaming of 360° videos through a server-client system. The streaming system was built to understand the intricacies of video streaming and testing our model. The system is divided into two parts:

1. **Server:** The server stores the video encoded in a certain format in different bitrates such that it is feasible for network transfer. The video frames cannot be directly transmitted over the network due to bandwidth constraints and different bitrates of different tiles. For each chunk, for each tile, the server would store it encoded in different bitrates. Whenever a client requests for some video chunk, given the bitrates of each of the tiles, the server simply returns the tile corresponding to the bitrate requested.

2. **Client:** The client streams the video to the user, collects the viewports for t seconds (t is the chunk duration; in our case, t=1) in the back-end, applies the model to predict the bitrates for the following chunk and requests the same from the server. Upon receiving the requested chunks for each tile, they are decoded and streamed by the client.

4.4.1 Video Encoding

The video first needs to be compressed in a content representation format for efficient storage and streaming. We have used the High Efficiency Video Coding format, also known as *HEVC* or *H.265* for video coding. This format provides better compression at the same video quality, thus saving storage space and better quality for the same bitrate, compared to other previous encoding formats using slightly higher computation costs. HEVC trades off computation cost for lower storage [31] and it is beneficial, since computation is one-time, however lower file size has the benefit during the transfer of data over the network.

HEVC aims towards providing twice the compression efficiency than the previous version H.264, thus reducing the file size for a certain segment, hence providing much better quality for the same bitrate [32]. The difference between HEVC and its previous versions lies in its ability to use extra computation to discover larger redundant areas in the same frame and within multiple frames, hence reducing the file size. The HEVC encoder uses the concept of motion vector to detect the redundant regions in the video frames.

Motion Constrained Tile Set [33, 34] or MCTS is a mechanism of encoding used in tiling-based streaming of videos, where the motion vector is restricted within the tile that it belongs to itself. Thus, a tile can reference to regions only within the same tile. Hence, any redundant areas will be encoded within the same tile, so that each tile can be encoded, transmitted and decoded independently and there are no dependencies among different tiles. In case of viewport adaptive 360° video streaming, since each tile chunk is sent at a different bitrate based upon the viewports in that chunk, if there are dependencies between different tiles, the chunks cannot be encoded and decoded separately and hence, cause problems. Hence, we use motion constrained tiling in this case.

To convert the videos to HEVC standard, we use *'Kvazaar'* [35], an open source HEVC/H.265 encoder. It is used to encode the video given the bitrate, number of tiles and the number of frames per second. It generates a raw HEVC bitstream which contains all the encoded information for various tiles. To divide the encoded bitstream spatially

and package it, we use the MP4Box Tool by *GPAC* [36] to generate an MP4 file containing a base track of parameter sets and one track for each tile.

4.4.2 Generating DASH Segments

The output file obtained from the above step is then fed to MP4Box to generate DASH Segments. DASH (Dynamic Adaptive Streaming over HTTP)[14] segments are a sequence of small HTTP segments, each containing a certain playback duration of the video. DASHing videos is needed for streaming media content over conventional HTTP server-client architecture. MP4Box can generate segments of one second and since we have used the motion constrained tile based encoding, the output has segments for multiple tiles, each in different size. The size corresponds to the bitrate of the tile. These DASH segments can be generated for multiple bitrates (different qualities) and stored on the server side. These segments are in the m4s format.

DASHing also creates MPD (MPEG-DASH) files which hold the information on the various streams and their associated bandwidths. At a higher level, the MPD file basically stores the meta-data of the location of video segments along with video meta-data in XML-based format.

4.4.3 Streaming 360° videos

Once the DASH Segments are generated and stored on the server side, the 360° videos can be streamed using *MP4Client* [36], which requires the server address and the location of the MPD file to stream the video. MP4Client is an easily configurable multimedia player created as a part of the GPAC that can be used to stream the 360° videos easily.

The main components of MP4Client are:

1. Input, demuxing and decoding filters as defined as part of GPAC
2. The Compositer module, which is responsible for media composition and interactivity in the video
3. The aout filter for audio output processing

To test the viewport adaptive streaming, the PARIMA model was written in the Compositor module of the MP4Client. The video streaming system was tested using two computers: one acted as a server with all the video data and the other acted as a client which streamed the video.

Chapter 5

Evaluation Testbed

This chapter will abridge the evaluation of the algorithm on various datasets. The evaluation strategy will follow a two-fold approach, that is, assess the accuracy of viewport prediction and summarize the Quality of Experience (QoE) metric. First, we will give a brief description of the datasets on which we have evaluated our algorithm, baseline against which we have presented the results and then discuss about the metrics.

5.1 Datasets

For evaluation of the various algorithms implemented, we use two popular datasets containing several 360-degree videos of different categories along with head tracking log. The first dataset (ds1) [4] includes five videos freely viewed by 59 users each. Every user has watched 70s of each of the videos during which their head movement logs have been tracked using a Head Mounted Display (HMD). The second dataset (ds2) [37] has 9 popular videos watched by 48 users. The average duration of the head tracking logs is 164 seconds, with a minimum of 60 seconds and a maximum of 655 seconds. Essentially, we have datasets corresponding to 727 cases of users watching 360° videos. Each trace of the head tracking logs for both the datasets consists of the user head position in terms of unit quaternions $(q_0, q_1, q_2, q_3) : q = (q_0, q_1\hat{\mathbf{i}} + q_2\hat{\mathbf{j}} + q_3\hat{\mathbf{k}})$ along with the frameID and timestamp.

From the head tracking logs, viewport and the viewport-specific tile are derived according to the algorithm suggested by Nguyen [10]. Given a head orientation represented as a quaternion, the head orientation vector u can be derived by applying a counter-clockwise rotations along a specific axis on its reference unit vector v. We have used the python library '*pyquaternion*'[38] for this transformation. Thus from the head orientation vector $u = (x, y, z)$ on the 3D sphere, we can easily convert it to get the pixel coordinates

using the formulae:

$$a = (\frac{1}{2} + \frac{\theta}{2\pi}) * \mathcal{W}$$

$$b = \left(1 + \frac{\phi}{\pi}\right) * \mathcal{H}$$

where a and b are the longitude and latitude positions in the equirectangular frame, θ and ϕ are the horizontal and the vertical angles of the head orientation vector u in 3D space (following our convention), $\mathcal{W}$ and $\mathcal{H}$ are the width and height of the target equirectangular frame.

Details of the dataset statistics can be found in Appendix A.

5.2 Baselines

The baselines against which we have evaluated our model are listed in this section along with a brief explanation of each of them.

5.2.1 PanoSalNet

Saliency Map based viewport prediction has been explored in PanoSalNet [7] where the authors have trained a Deep Convnet (DCNN) architecture to understand the salient portions of each panoramic frame of a 360-degree video. The main intuition behind constructing the saliency map of an image is that the user head movement and visual attention depends mostly on specific portions of a video that are termed as salient, a claim concurrent to ours. Coupled with saliency map, the authors have integrated user head tracking history for the ultimate head movement prediction via an LSTM architecture.

5.2.2 Non-Adaptive Bitrate Allocation (NABA) Model

The primary goal of our video streaming model was to improve the Quality of Experience of the user by providing higher bitrate to the tiles corresponding to the viewport and hence, higher quality to those tiles. Hence, an important baseline to judge our model is a non-adaptive 360° video streaming model. Under this streaming model, there is no viewport-adaptation and hence, bitrate is allocated to all the tiles in a uniform manner regardless of the viewport. In general, if B is the bitrate of streaming and the video is spatially divided into M x N tiles, then the bitrate allotted to a tile will be B/(M x N). Any adaptive streaming model would be better only if the QoE obtained for the video is better than non-adaptive streaming model.

5.3 Evaluation Metrics

5.3.1 Prediction Metrics

The metrics defined below is a measure of the prediction algorithms used in the evaluation.

Mean Absolute Error (MAE)

Let (x, y) be the pixel level actual viewport as obtained from user head movement dataset, and (x', y') be the pixel level viewport predicted by an algorithm. Mean Absolute Error is the average absolute error between the x and y predicted pixels and the actual pixels. We have separate metrics to log the error in x-coordinate and y-coordinate. Mathematically, it is defined as

$$MAE_x = \frac{\sum_{f=1}^{n_f} |x_f - x'_f|}{n_f}$$

We have reported an averaged MAE value over all frames and users for a particular video.

Manhattan Tile Error

Let (x, y) be the actual tile index of the viewport, that is, the tile in which the actual viewport lies, and (x', y') be the predicted tile index. Manhattan Tile Error is the average manhattan distance between (x, y) and (x', y'), that is, $|x - x'| + |y - y'|$ for all frames in a video.

It is to be noted that in a 360-degree setting, minimum manhattan distance between two tiles can be as a result of wrapping around either in the vertical, horizontal or in both the direction. For example, in a scenario of 8×8 tiling, with actual viewport tile index as $(0, 1)$ and predicted viewport tile index as $(1, 7)$, the Manhattan Tile Error will be $|0 - 1| + |(8 - 7) - 1| = 3$, which included wrapping horizontally. Secondly, we have taken Manhattan Tile Error as 0 if the predicted viewport lies within video player FoV of the actual viewport.

Matrix Error

Let $\mathcal{A}$ and $\mathcal{B}$ be mn matrices where $m \times n$ is the tiling order. Let the actual tile index of the viewport be (x, y) and predicted tile index be (x', y'). The matrices are defined as:

$$\mathcal{A}(x, y) = 1, \mathcal{A}(i, j) = 0 \ \forall i \neq x, j \neq y$$

$$\mathcal{B}(x', y') = 1, \mathcal{A}(i, j) = 0 \ \forall i \neq x', j \neq y'$$

Matrix Error between two matrices $\mathcal{A}$ and $\mathcal{B}$ is defined as:

$$d = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}(\mathcal{B}(i,j) - \mathcal{A}(i,j))^2}$$

We report the average Matrix Error of all the frames in a video.

Accuracy

Accuracy is the fraction of the number of frames in which the predicted viewport was in the video player FoV of the actual viewport over the total number of frames in the video. Simple, it is the fraction of the number of frames having Manhattan Tile Error=0 over the total number of frames.

$$\text{Acc} = \frac{\text{\# Frames with Manhattan Tile Error} = 0}{\text{Total Frames}}$$

5.3.2 QoE Metrics

User perceived quality is measured in deterministic fashion using several QoE metrics that we define which empirically evaluates the performance of our approach.

1. The first of the QoE metric (Q_1) will be the average bitrate consumed by the user. In essence, it denotes the quality of the video perceived by the user. For each chunk of frames, we calculate the normalized sum of bitrates of the tiles that lies in the viewport. Let there be $\mathcal{X} \times \mathcal{Y}$ number of tiles in the video with a media player viewport dimension as $P_w \times P_h$. Mathematically, for chunk c, Q_1^c is denoted as

$$Q_1^c = \frac{1}{n_c}\sum_{i=1}^{f_c}\left(\frac{\sum_{P_w \times P_h} a_{x,y}^i B_{x,y}^c}{tiles(P_w) \times tiles(P_h)}\right)$$

   where, f_c is the number of frames in chunk c, or equivalently it is *fps*duration_of_chunk*. Each tile can be represented using cartesian coordinates where $x \in \mathcal{X}, y \in Y$. $B_{x,y}^i$ elicits the bitrate for $(x,y)^{th}$ tile in chunk c and $a_{x,y}^i$ is an indicator variable which becomes 1 if tile (x,y) is in the viewport of i^{th} frame of chunk c. $tiles(P_w) \times tiles(P_h)$ amounts to the total number of tiles present in the viewport.

   n_c is the normalizing constant which is calculated by the sum of distinct tiles that are in the viewport for chunk c.

2. The second QoE metric (Q_2) is a measure of the variation of quality within the viewport for each frame. This is an important measure while maximizing QoE since

we want to have a minimum variation among the qualities of various tiles in the viewport. For chunk c, we denote this metric Q_2^c as

$$Q_2^c = \frac{1}{n_c} \sum_{i=1}^{f_c} StdDev\{B_{x,y}^c : \ x \in tiles(P_w), y \in tiles(P_h)\}$$

3. The third QoE (Q_3) is similar to Q_2 but it captures the variation of quality among different frames for a chunk.We would like to minimize the amount of variation of quality from a viewport of frame f_1 to a viewport of frame f_2. For chunk c, we denote this metric Q_3^c as

$$Q_3^c = \frac{1}{n_c} StdDev\{\frac{\sum_{P_w \times P_h} a_{x,y}^i B_{x,y}^c}{tiles(P_w) \times tiles(P_h)} : a_{x,y}^i = 1; \ \forall i \in f_c, \ x \in \mathcal{X}, y \in \mathcal{Y}\}$$

4. Variation of quality across different chunks also impacts QoE, which forms the third component of the QoE metric (Q_4). Thus, it is important to quantify the amount of variation of quality from one chunk to the immediate next chunk. For any arbitrary chunk c, Q_4^c can be mathematically expressed as

$$Q_4^c = |Q_1^c - Q_1^{c-1}|$$

The final QoE metric is an aggregation of all the above described components, which is expressed as:

$$Q = \sum_{c=1}^{C} (Q_1^c - \eta_1 Q_2^c - \eta_2 Q_3^c) - \eta_3 \sum_{c=2}^{C} Q_4^c$$

where C is the total number of chunks that has been formed, and $\eta_1, \eta_2 \text{and} \eta_3$ are hyper-parameters which specifies the amount of importance we give to each of the QoE metric.

5.4 Evaluation Setting

- In our experiments, each video is temporally segmented into chunks of 1 second duration, which are further spatially segmented into 8 x 8 tiles, thus forming a total of 64 tiles. We have exposited a study in the next section showing our choice of temporal chunk duration.

- Video player Field of View(FoV) is chosen to be of the dimension of 600×300 [39, 40] for experimentation.

- For the horizontal dimension, we have chosen the ARIMA model to be of order (2,1,1) while for the vertical dimension, the order is (3,1,0). By examining the Autocovariance Function (acf) and the Partial Autocovariance Function (pacf), we have finalized the orders of the ARIMA model.

- For the Passive Aggressive Regression model, the hyperparameters that we have chosen are: i) $C = 0.01$, ii) $\epsilon = 0.001$, and iii) learning rate = 0.001.

Chapter 6

Evaluation Results

This chapter discusses the results of the evaluation carried out over the duration of the project. The chapter is organised as follows:

1. Evaluation for the best regression model for predicting one frame at a time
2. Finding the optimal chunk size for video streaming
3. Analysis of performance of various models and comparison with baseline Panosalnet
4. Comparison of performance of the models with non-adaptive video streaming to ensure better QoE

6.1 Prediction For One Frame: Regression Models

The first set of experiments were performed to judge the best regression model that can be used among different possibilities for predicting the viewport for the next frame given the object trajectories and the viewport of the previous frame. The reason for trying regression models over other machine learning models has been discussed in section 4.2.1. We first try predictions for only one frame because if a model cannot predict well for one frame, then it won't be able to give reliable predictions for a chunk of frames. For this experiment, we built a Java application for viewport data collection over a 360° video. s

6.1.1 Viewport Data Collection

An android application was created which was responsible for playing the video and capturing the viewport throughout the length of the video. The android application uses 'Google VR SDK' API through which the video can be played in VR mode. The

application is built in order to detect the viewport of the users. It does so using the gyroscopic fluctuations while playing the video. The gyroscopic sensor of the phone triggers the EventListener object of the Java application, which periodically saves the gyroscopic data into a file. A screenshot of the application is provided in Appendix B.

Gyroscopic data provides angular velocity in each Cartesian direction when the device is moved. Thus, from the data, we can get the orientation of the view as well as the position of the view in a spherical space. One can easily use the trapezoidal rule of integration to calculate the angular displacement when the device is moved.

Let the initial center of the viewport be $(x_{init}, y_{init}, z_{init})$. After time t, we record the gyroscopic data for three axes as (α, β, γ). Therefore the new center of the viewport at time t can be calculated as:

$$(x_{new}, y_{new}, z_{new}) = (x_{init} + \alpha t, y_{init} + \beta t, z_{init} + \gamma t)$$

Thus, the new viewport in the spherical space now becomes $(x_{new}, y_{new}, z_{new})$.

6.1.2 Regression Models

Several regression models were experimented to determine the appropriate model for predicting future viewport when the viewport data can change abruptly. The models were tuned so as to predict the viewport of only the next frame, which was then later generalized to a prediction window. The different models that were used comprised of:

- **Linear Regression:** A simple linear regression model that takes the positions of the objects and the previous viewport as input and produces the predicted next viewport as the output.

- **Passive-Aggressive Regression:** It was already discussed in Section 4.2.1. However, we need to make a minor trivial change to the model to predict for only the next frame, that is, have a prediction window of 1.

- **Box-Cox Regression:** Box-Cox power transformation is often used to modify the distributional shape of the response variables so that the residuals are more normally distributed. The transformation often proves to be effective for stabilizing variance and make the distribution more 'normal-like'. In our method, we apply Box-Cox transformation on the target variable, that is, the viewport information.

6.1.3 Observations

In this section, we will evaluate the performance of the regression models on a 360° video. The video involved 3 moving prime objects. Viewers were asked to look at the video through the android application that captured the angular velocity of the device. We have used *creme* [41], an incremental machine learning library to build our model and evaluate the performance.

The hyper parameters used for the different models are as follows:

- Linear Regression: SGD optimizer with $\eta = 0.0005$
- PA Regression: $C = 0.01, \epsilon = 0.001$
- Box-Cox Regression: $\lambda = 0.8$, $\eta = 0.0001$

To compare different models of regression, we analyse them at pixel level and hence, we plot the MAE values against the frames viewed by the user (Figure 6.1). We also plot time series graphs of the actual and predicted viewport with respect to the frame number (Figure 6.2). Table 6.1 and 6.2 tabulates the performance of each model on the video.

	Linear Regression	**PA Regression**	**BoxCox Regression**
User 1	155.125	18.867	193.271
User 2	157.766	16.791	215.367
User 3	182.246	30.038	256.309
User 4	230.048	70.362	304.074

Table 6.1: *Final MAE value (after trained on the full video) for different regression models.*

	Linear Regression	**PA Regression**	**BoxCox Regression**
User 1	10.76%	1.30%	13.42%
User 2	10.96%	1.16%	14.96%
User 3	12.66%	2.08%	17.78%
User 4	15.96%	4.88%	21.12%

Table 6.2: *To understand the prediction qualitatively, we judge the MAE as percentage of the maximum possible error* $(width + height)/2$

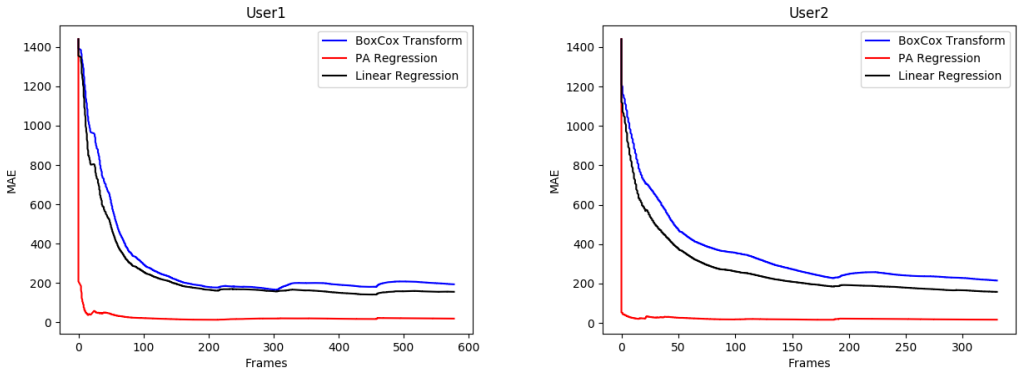


Figure 6.1: *Comparison of MAE for different regression models for two users*

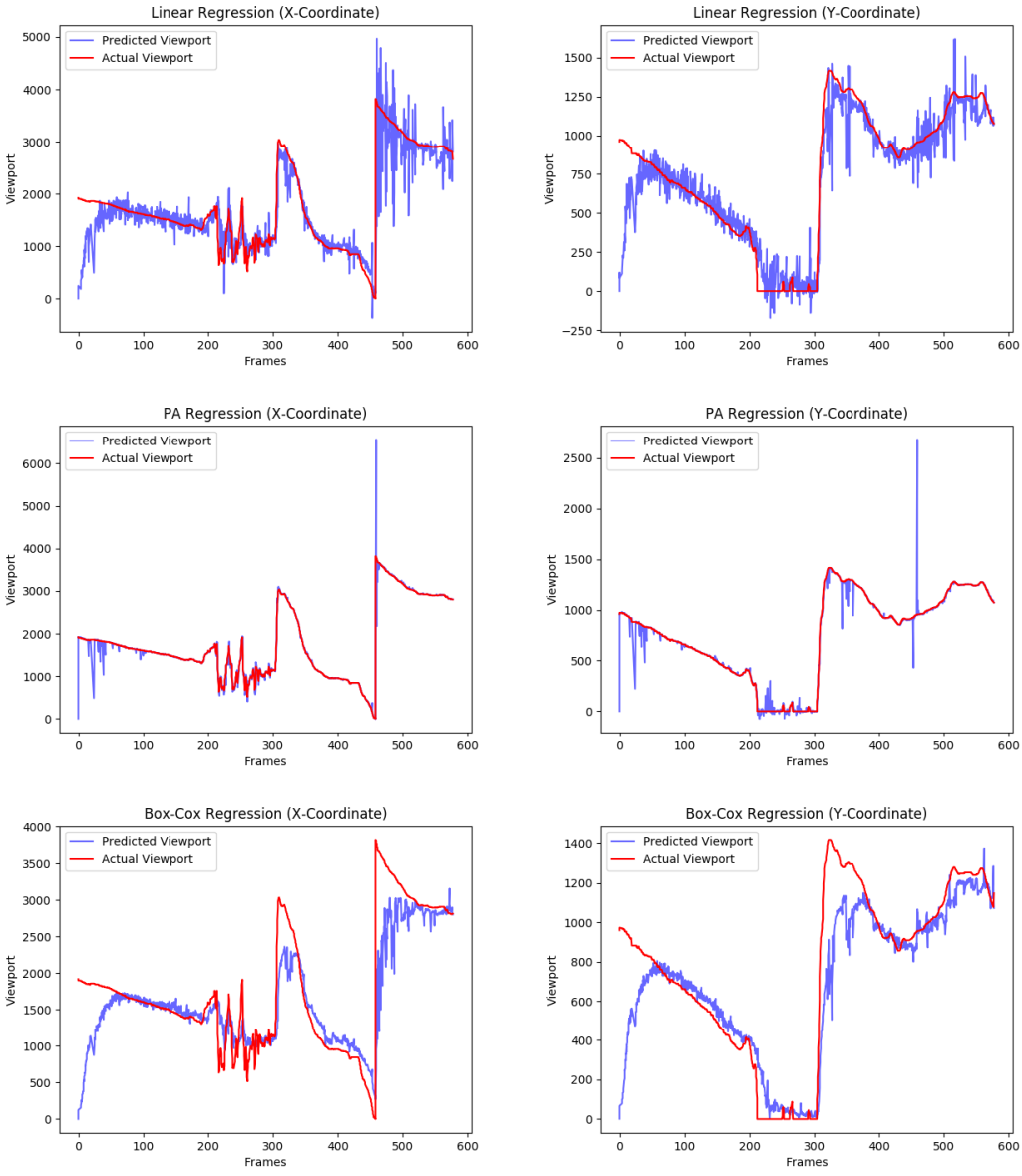


Figure 6.2: *Plot of actual and predicted viewports with respect to the frames for User 1*

It is evident from the results that Passive-Aggressive Regression is the best regression model among the rest. This also supports the statements from [42] that these are online learning algorithms that have been found to be superior than many other learning algorithms.

6.2 Predicting Viewport for Multiple Frames

The above experiments were to find the model that fits best for predicting the next frame. However, predicting the viewport for each future frame and pinging the server to deliver the next frame with a certain quality is not feasible. This is because there is a lag between the communication of server and client and the prediction time. Hence, the video will buffer after every frame. Thus, the next aim is to predict the viewports in a prediction window of chunk of size t. Such a setup would make the streaming feasible, since the client will buffer an entire chunk of video comprising of multiple frame by the time it gets a new chunk.

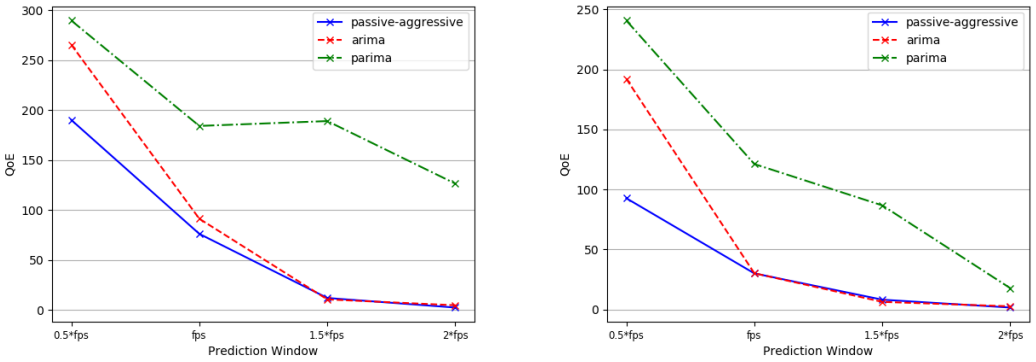
6.2.1 Analysis of Optimal Chunk Size

Studies like Flare [18] have analysed that 1 second chunk duration is optimal for viewport-adaptive 360° video streaming. Choosing the optimal chunk size essentially involves a trade-off between the prediction accuracy and the buffering time. If we keep a small chunk size, the prediction of viewport for the frames in that chunk will be more accurate, however the prediction and data-transfer time would have to be fast. For larger chunk sizes, on the other hand, we might suffer from poor accuracy. We analysed the three models discussed in Section 4.2 for four different chunk sizes: 0.5 seconds (chunk size: fps/2 frames), 1 second (chunk size: fps frames), 1.5 seconds (chunk size: 3fps/2 frames) and 2 seconds (chunk size: 2fps frames).

We obtained the results for two videos from dataset 1, namely paris and timelapse, and plotted the average QoE for the 59 users for different learning models with respect to chunk size. The plots for the same have been shown in Figure 6.3.

6.2.2 Observations

It was observed that for almost all cases, the QoE reduces as we increase the chunk size, which was the expected result, since we assume in the experiment that we have sufficient bandwidth. QoE only accounts for the bitrate distribution and network adaptation is not considered for streaming.



(a) *Video 'paris' of dataset 1. fps=60*

(b) *Video 'timelapse' of dataset 1. fps=30*

Figure 6.3: *The impacts of prediction window on the QoE of the head movement prediction models*

However, for practical streaming purposes, it is essential to note that choosing chunk size as fps (or 1 second) is better than fps/2 (or 0.5 seconds) due to the following reasons:

1. **Cannot DASH Segments of less than 1 second:** Every video encoding algorithm generates video segments of at least 1 second. Smaller duration DASH segments cannot be created.

2. **Bitrate and Compression Inefficiency:** If we have a smaller chunk size, then less number of redundant areas would be discovered within that chunk, since more number of frames means more possibility of combining redundant data. Also, every segment is accompanied by additional meta-data. Smaller chunk size means larger number of chunks, less compression of redundant areas and more meta-data. Hence, the total size of two 0.5 second chunks would be larger than one 1 second chunk. Thus, for the same bitrate, smaller chunks would be streamed at comparatively poor quality.

3. **Prediction and Network Transfer Time:** Practical streaming systems may not have sufficient bandwidth and hence, prediction of viewport and transfer of data from server to client may take time more than 0.5 seconds sometimes.

Through the above arguments, we claim that chunk size of 1 second duration is optimal in general.

Topic	# Objects	Metric	PA	ARIMA	PARIMA	PanoSalNet
paris	8	Tile Error	1.434	0.008	0.612	1.237
		Matrix Error	1.098	0.412	1.097	1.112
		QoE	76.08	91.20	**184.02**	72.02
timelapse	61	Tile Error	0.825	0.008	0.685	1.481
		Matrix Error	1.095	0.456	1.139	0.996
		QoE	30.02	30.15	**121.45**	38.69
venice	14	Tile Error	0.994	0.006	0.353	1.124
		Matrix Error	1.130	0.387	1.110	0.995
		QoE	47.36	229.36	**303.87**	92.49
diving	41	Tile Error	0.524	0.007	0.337	1.341
		Matrix Error	0.954	0.401	1.235	1.001
		QoE	219.69	204.03	**234.58**	126.55
roller	66	Tile Error	0.495	0.009	0.234	0.928
		Matrix Error	0.958	0.382	0.827	0.998
		QoE	173.54	137.18	**384.22**	112.45

Table 6.3: *Comparing Metrics for videos in ds1.*

6.3 Model Comparison

From the previous section, we have said that the optimal prediction window that must be used should be 1 second (fps frames). We have used this predicition window for all the experiments henceforth. However, with a variety of model at our disposal, we need to choose the appropriate model that would qualify our evaluation. Hence, this section exposits a comparative study between the different algorithms that we have stated earlier. A comprehensive evaluation report is tabulated in Table 6.3 and 6.4. The results that have been shown are averaged over all the users for a particular video.

We observed that PARIMA consistently exhibits a superior QoE than the other models for almost all the videos. Passive-Aggressive Regression tends to show extremely high tile error, and even overshoot in some cases. ARIMA model displayed extremely low tile error values and the tile error for PARIMA is low as well. The steep difference in the QoE between ARIMA and PARIMA models is attributed in the bitrate allocation scheme, because of which ARIMA had large difference in the bitrates of the predicted viewport and non-viewport regions, whereas bitrate allocation for PARIMA was smoother over various chunks, leading to lower Q_4 values. The matrix error which is a measure of the prediction

error is comparable for PARIMA and PA for ds1 but is minutely deviated from the ARIMA model for the datasets. However, the baseline PanoSalNet consistently performs worse than ARIMA or PARIMA in all the metrics, thus supporting the experimental results.

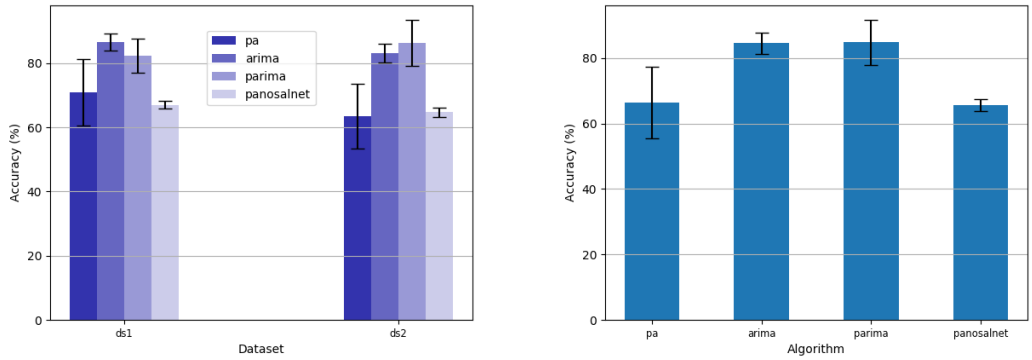
Topic	# Objects	Metric	PA	ARIMA	PARIMA	PanoSalNet
Vid0	41	Tile Error	2.612	0.025	0.144	0.458
		Matrix Error	1.338	0.354	0.595	0.898
		QoE	26.84	184.76	**225.34**	150.42
Vid1	77	Tile Error	1.796	0.027	0.155	1.995
		Matrix Error	1.298	0.368	0.674	0.938
		QoE	32.98	143.85	**172.95**	79.88
Vid2	92	Tile Error	3.358	0.014	0.149	1.764
		Matrix Error	1.381	0.395	0.671	0.899
		QoE	13.23	**346.01**	337.70	173.76
Vid3	25	Tile Error	45.306	0.025	0.133	2.141
		Matrix Error	1.264	0.333	0.556	0.789
		QoE	48.09	390.82	**403.42**	128.47
Vid4	40	Tile Error	2.616	0.033	0.177	1.217
		Matrix Error	1.345	0.483	0.770	0.899
		QoE	4.71	52.75	**98.56**	71.45
Vid5	397	Tile Error	1.483	0.035	0.178	1.491
		Matrix Error	1.228	0.425	0.715	0.937
		QoE	30.77	**300.61**	291.32	150.48
Vid6	1105	Tile Error	2.758	0.007	0.178	0.823
		Matrix Error	1.414	0.229	0.715	0.966
		QoE	39.93	184.66	**219.32**	191.035
Vid7	166	Tile Error	2.342	0.012	0.175	1.495
		Matrix Error	1.323	0.408	0.731	0.813
		QoE	19.12	195.79	**198.97**	168.612
Vid8	77	Tile Error	1.707	0.016	0.104	1.236
		Matrix Error	1.302	0.329	0.600	0.809
		QoE	42.58	477.11	**498.71**	203.824

Table 6.4: *Comparing metrics for videos in ds2.*

We can note a fact from the tables (Table 6.3 and 6.4) that the PARIMA model

provides consistent results when there are less object (ds1, topic 'paris') as well as with large number of objects (ds2, topic 'Vid6'), thus showing that the PARIMA model is robust to the number of objects present in the video. However, it is to be noted that the number of objects displayed in the tables is an empirical one and is calculated using the Object detection algorithm proposed in Sections 4.1.2 and 4.1.4.

Figure 6.4 plots the accuracy of the various models on the two datasets. The accuracy values are tabulated in Table 6.5. Accuracy is defined as in Section 5.4.1. We have reported the average accuracy over all the videos and users for a particular dataset.We observe that the accuracy of PARIMA and ARIMA are comparable in both the datasets while PanoSalNet performs below par with respect to PARIMA. This again supports our claim that viewport depends on the object trajectory and hence the viewport contents.



(a) *Overall Accuracy of different algorithms for both the datasets.*

(b) *Combined overall Accuracy of bothe the datasets*

Figure 6.4: *The accuracy of the proposed models ARIMA and PARIMA are comparable in both the datasets and beats the baseline PanoSalNet comprehensively.*

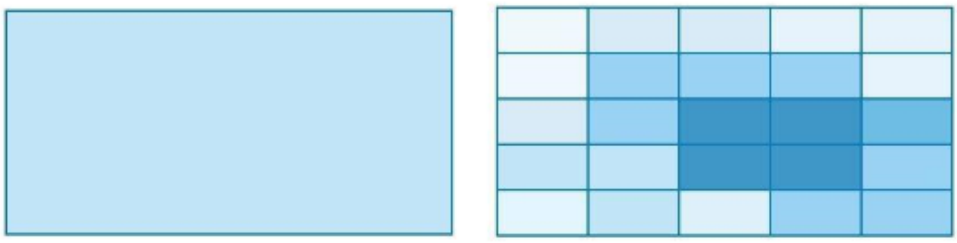
Dataset No.	PA	ARIMA	PARIMA	PanoSalNet
ds1	70.92%	86.6%	82.32%	67.04%
ds2	63.53%	83.2%	86.26%	64.77%
ds1+ds2	66.37%	84.51%	84.77%	65.64%

Table 6.5: *Accuracy of models for the different datasets.*

6.4 Viewport Adaptivity: Comparison with NABA

In the final set of experiments, we compare the adaptive vs non-adaptive bitrate allocation scheme and elicit the conclusion where one performs better than the other.

6.4.1 Rate Adaptive Tiles



(a) *Non Adaptive Bitrate Allocation. Download the full video*

(b) *Pyramid Bitrate Allocation. Download tiles according to rate*

Figure 6.5: *Example illustrating the difference between two schemes of bitrate allocation and downloading the video in different resolution for the two schemes*

As stated in Section 3.4, bitrate of a video refers to the amount of data that is streamed over a network in a second. Hence, for fixed bitrate (Eg., 5Mbps), the network can transfer a 5Mb of data over 1 second. Hence, irrespective of the frame size of a video, a fixed amount of data can be transferred in a chunk of 1 second. Thus, while streaming a video using non-adaptive bitrate allocation scheme (Figure 6.5(a)), all the tiles of the video gets equal distribution of rates and since all rates of all the tiles must add up to a maximum allowable bitrate amount, each tile gets a significantly lower amount. For example, for 5Mbps bandwidth and a 360°video with 64 tiles, each tile gets a rate of 5/64 Mbps.

On the contrary, if we use an adaptive bitrate based on viewport prediction (Figure 6.5(b)), each tile would get a weighted rate. Thus, a tile having a maximum probability of being in the viewport will get a higher share than the tiles that are far away from the predicted viewport. Thus, such a distribution of rate among the tiles would increase user experience and improve QoE.

6.4.2 Observations

Bitrate Allocation	paris	timelapse	venice	roller	diving
Pyramidal (PARIMA)	184.02	121.45	303.87	234.58	173.54
NABA	47.41	38.34	53.15	66.77	53.91

Table 6.6: *Average QoE for all users of each video in ds1*

Bitrate Allocation	Vid0	Vid1	Vid2	Vid3	Vid4	Vid5	Vid6	Vid7	Vid8
Pyramidal (PARIMA)	225.34	172.95	337.70	403.42	98.56	291.32	191.32	195.97	498.71
NABA	170.65	157.73	154.88	176.76	93.86	161.93	162.11	148.02	169.73

Table 6.7: *Average QoE for all users of each video in ds2*

To demonstrate the above result, we have experimented with two bitrate allocation schemes. NABA refers to the non-adaptive bitrate allocation scheme, while for the adaptive rate allocation, we have chosen the PARIMA model. As already mentioned, we have used a pyramid based rate allocation scheme. Table 6.6 and 6.7 tabulates the average QoE of all the users for videos of particular datasets.

We observe that Pyramidal allocation scheme defeats NABA in almost all the videos, and hence we can say that in general setting, adaptive bitrate scheme performs better than NABA.

Chapter 7

Conclusion and Future Work

7.1 Conclusions from the Experiment

IN summary, we have developed an end-to-end streaming platform for 360°videos that includes the prediction of viewport followed by pyramid-based rate adaptation of tiles based on the predicted viewport. From the extensive experiments conducted, several conclusions have surfaced which are as follows:

1. Using the results tabulated, we concluded that the viewport of the user depends upon the previous viewport as well as the positions of the prime objects in the video. Since it is nearly impossible to obtain exact object trajectories, we used an approximate method, which was observed to work well.

2. PARIMA model leverages the benefits of the time series model as well as the Passive-Aggressive Regressor. ARIMA model can capture the locality of viewport better since it is fed with data of only the previous chunk, while the information of object trajectory is better captured by the Passive Aggressive Regressor, since it can adapt quickly. With the experiments performed, we can observe that PARIMA has the highest QoE value and beats the baseline comprehensively.

3. A prediciton window of 1 second (fps frames) is optimal owing to the fact that compression efficiency is maintained and network transfer time is optimal. A smaller chunk size would make a video buffer due to network transfer time.

4. Adaptive bitrate allocation exhibits a higher QoE than NABA, owing to the fact that it brings only those portions of the video with a higher bitrate which is predicted to be in the viewport for the next chunk. Hence, this increases the overall quality of the video as seen by the user.

7.2 Future Work

We have developed a viewport adaptive streaming platform where the next frames are transferred with a quality based on the viewport prediction.However, there lies an underlying assumption that network bandwidth will remain constant and would not under perform. Thus our model is agnostic towards network inconsistencies.

Hence, as a next step and a sane direction would be to predict the network inconsistencies and allocate the tile rate based on it. Thus as a future work, we can combine the viewport based adaptive rate and network based adaptive rate to get a better understanding of the network over which the video is to be transferred and hence improve QoE. We also plan to further validate our work using larger studies related to live streaming large datasets.

Saliency map can be used as well along with object trajectories to get a better representation of the video. This would help in decreasing prediction accuracy. Instead of pixel level prediction, probability based prediction can be also be a possible direction.

Appendices

Appendix A

Dataset Statistics

Video Name	Content Description & Expected Focus of Attention	Spatial Resolution	Frame Rate	Bit Rate	Start Offset
Diving	Diving scene. Slowly moving camera, no clear horizon. No main focus expected within the sphere.	3840×2048 pixels	29.97 fps	19604 kbps	40 s
Rollercoaster	Rollercoaster. Fast moving camera fixed in front of a moving roller-coaster. Strong main focus following the rollercoaster trail.	3840×2048 pixels	30 fps	16075 kbps	65 s
Timelapse	Timelapse of city streets. Fixed camera, clear horizon with a lot of fast moving people/cars, many scene cuts. Focus expected along the equator line.	3840×2048 pixels	30 fps	15581 kbps	0 s
Venice	Virtual aerial reconstruction of Venice. Slowly moving camera. No main focus expected within the sphere.	3840×2048 pixels	25 fps	16101 kbps	0 s
Paris	Guided tour of Paris. Static camera with some smooth scene cuts. Focus expected along the equator line.	3840×2048 pixels	60 fps	14268 kbps	0 s

Table A.1: Description of the YouTube 360-degree videos used in the first dataset described in Section 5.1(ds1).

Video No.	Video Name	Category	Spatial Resolution	Frame Rate	Bit Rate
0	Conan360-Sandwich	Performance	2560×1440 pixels	29 fps	5619 kbps
1	Freestyle Skiing	Sport	2560×1440 pixels	30 fps	8491 kbps
2	Google Spotlight-HELP	Film	2560×1440 pixels	30 fps	7868 kbps
3	Conan360-Weird Al	Performance	2560×1440 pixels	25 fps	5672 kbps
4	GoPro VR-Tahiti Surf	Sport	2560×1440 pixels	29 fps	9317 kbps
5	The Fight for Falluja	Documentary	2160×1080 pixels	29 fps	4948 kbps
6	360 Cooking Battle	Performance	2560×1440 pixels	25 fps	5423 kbps
7	LOSC Football	Sport	2560×1440 pixels	25 fps	6083 kbps
8	The Last of the Rhinos	Documentary	2560×1440 pixels	29 fps	4236 kbps

Table A.2: Description of the 360-degree videos used in the second dataset described in Section 5.1(ds2).

Appendix B

Android Application for Viewport Data Collection

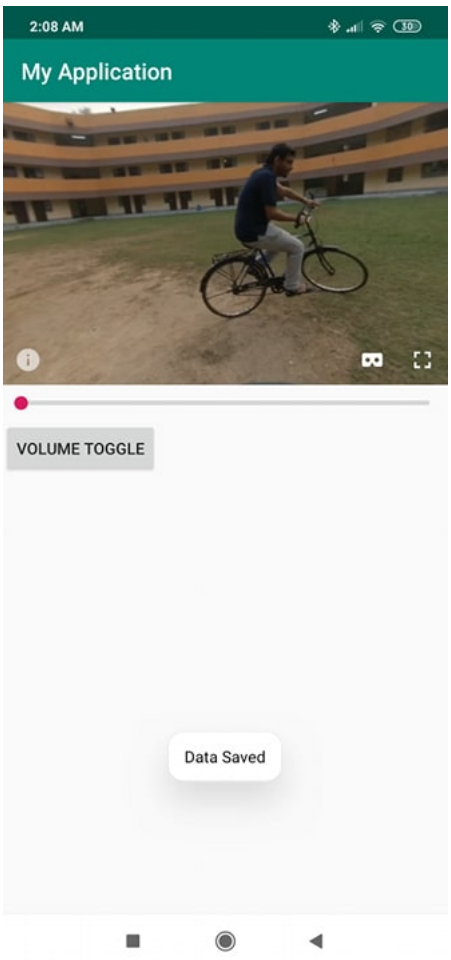


Figure B.1: *Android Application that can collect the viewport data using gyroscopic metrics. Data gets saved incsv format which can be further processed to get the pixel level viewport*

References

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

[2] Facebook end-to-end optimizations for dynamic streaming. https://engineering.fb.com/video-engineering/end-to-end-optimizations-for-dynamic-streaming/.

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

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

[5] Ching-Ling Fan, Jean Lee, Wen-Chih Lo, Chun-Ying Huang, Kuan-Ta Chen, and Cheng-Hsin Hsu. Fixation prediction for 360 video streaming in head-mounted virtual reality. In *Proceedings of the 27th Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 67–72, 2017.

[6] Sohee Park, Arani Bhattacharya, Zhibo Yang, Mallesham Dasari, Samir R Das, and Dimitris Samaras. Advancing user quality of experience in 360-degree video streaming. In *2019 IFIP Networking Conference (IFIP Networking)*, pages 1–9. IEEE, 2019.

[7] Anh Nguyen, Zhisheng Yan, and Klara Nahrstedt. Your attention is unique: Detecting 360-degree video saliency in head-mounted display for head movement prediction. In *Proceedings of the 26th ACM international conference on Multimedia*, pages 1190–1198, 2018.

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

[9] 360 cinema on vimeo - the high-quality home for video. https://vimeo.com/channels/360vr.

[10] Anh Nguyen and Zhisheng Yan. A saliency dataset for 360-degree videos. In *Proceedings of the 10th ACM Multimedia Systems Conference*, pages 279–284, 2019.

[11] Alireza Zare, Alireza Aminlou, Miska M Hannuksela, and Moncef Gabbouj. Hevc-compliant tile-based streaming of panoramic video for virtual reality applications. In *Proceedings of the 24th ACM international conference on Multimedia*, pages 601–605, 2016.

[12] Xavier Corbillon, Gwendal Simon, Alisa Devlic, and Jacob Chakareski. Viewport-adaptive navigable 360-degree video delivery. In *2017 IEEE international conference on communications (ICC)*, pages 1–7. IEEE, 2017.

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

[14] Thomas Stockhammer. Dynamic adaptive streaming over http–: standards and design principles. In *Proceedings of the second annual ACM conference on Multimedia systems*, pages 133–144. ACM, 2011.

[15] 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.

[16] Mathias Almquist, Viktor Almquist, Vengatanathan Krishnamoorthi, Niklas Carlsson, and Derek Eager. The prefetch aggressiveness tradeoff in 360 video streaming. In *Proceedings of the 9th ACM Multimedia Systems Conference*, pages 258–269. ACM, 2018.

[17] Tam V Nguyen, Mengdi Xu, Guangyu Gao, Mohan Kankanhalli, Qi Tian, and Shuicheng Yan. Static saliency vs. dynamic saliency: a comparative study. In *Proceedings of the 21st ACM international conference on Multimedia*, pages 987–996, 2013.

[18] 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. ACM, 2018.

[19] Lan Xie, Xinggong Zhang, and Zongming Guo. Cls: A cross-user learning based system for improving qoe in 360-degree video adaptive streaming. In *2018 ACM Multimedia Conference on Multimedia Conference*, pages 564–572. ACM, 2018.

[20] 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.

[21] Mengbai Xiao, Chao Zhou, Yao Liu, and Songqing Chen. Optile: Toward optimal tiling in 360-degree video streaming. In *Proceedings of the 25th ACM international conference on Multimedia*, pages 708–716, 2017.

[22] Ching-Ling Fan, Wen-Chih Lo, Yu-Tung Pai, and Cheng-Hsin Hsu. A survey on 360 video streaming: Acquisition, transmission, and display. *ACM Computing Surveys (CSUR)*, 52(4):71, 2019.

[23] What is video bitrate and why it matters? https://filmora.wondershare.com/video-editing-tips/what-is-video-bitrate.html.

[24] Understanding bitrates in video files. https://help.encoding.com/knowledge-base/article/understanding-bitrates-in-video-files/.

[25] vrprojector. https://github.com/bhautikj/vrprojector.

[26] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[27] Simple object tracking with opencv. https://www.pyimagesearch.com/2018/07/23/simple-object-tracking-with-opencv/.

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

[29] Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. Online passive-aggressive algorithms. *Journal of Machine Learning Research*, 7(Mar):551–585, 2006.

[30] faculty.smu.edu. Augmented dickey-fuller unit root tests.

[31] HEVC(H.265): What is it and why should you care? https://blog.frame.io/2018/09/24/hevc-format-wars/.

[32] HEVC/H.265 explained. http://x265.org/hevc-h265/http://x265.org/hevc-h265/http://x265.org/hevc-h265/.

[33] Jangwoo Son, Dongmin Jang, and Eun-Seok Ryu. Implementing motion-constrained tile and viewport extraction for vr streaming. pages 61–66, 06 2018.

[34] Soonbin Lee, Dongmin Jang, JongBeom Jeong, and Eun-Seok Ryu. Motion-constrained tile set based 360-degree video streaming using saliency map prediction. In *Proceedings of the 29th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 20–24, 2019.

[35] Marko Viitanen, Ari Koivula, Ari Lemmetti, Arttu Ylä-Outinen, Jarno Vanne, and Timo D. Hämäläinen. Kvazaar: Open-source hevc/h.265 encoder. In *Proceedings of the 24th ACM International Conference on Multimedia*, 2016.

[36] Gpac, multimedia open source project. https://gpac.wp.imt.fr/.

[37] Chenglei Wu, Zhihao Tan, Zhi Wang, and Shiqiang Yang. A dataset for exploring user behaviors in vr spherical video streaming. In *Proceedings of the 8th ACM on Multimedia Systems Conference*, pages 193–198, 2017.

[38] Pyquaternion, python module for representing and using quaternions. http://kieranwynn.github.io/pyquaternion/.

[39] Vr video formats explained. https://360labs.net/blog/vr-video-formats-explained.

[40] Video player size. https://support.google.com/dcm/faq/6170528.

[41] Creme, online machine learning in python. https://creme-ml.github.io/.

[42] Ml algorithms addendum: Passive aggressive algorithms https://www.bonaccorso.eu/2017/10/06/ml-algorithms-addendum-passive-aggressive-algorithms/.