# PARIMA: Viewport Adaptive 360-Degree Video Streaming

Lovish Chopra*
Indian Institute of Technology
Kharagpur, India
lovishchopra98@gmail.com

Sarthak Chakraborty*
Indian Institute of Technology
Kharagpur, India
sarthak.chakraborty@gmail.com

Abhijit Mondal
Indian Institute of Technology
Kharagpur, India
am@abhijitmondal.in

Sandip Chakraborty
Indian Institute of Technology Kharagpur
Kharagpur, India
sandipc@cse.iitkgp.ac.in

## ABSTRACT

With increasing advancements in technologies for capturing 360° videos, advances in streaming such videos have become a popular research topic. However, streaming 360° videos require high bandwidth, thus escalating the need for developing optimized streaming algorithms. Researchers have proposed various methods to tackle the problem, considering the network bandwidth or attempt to predict future viewports in advance. However, most of the existing works either (1) do not consider video contents to predict user viewport, or (2) do not adapt to user preferences dynamically, or (3) require a lot of training data for new videos, thus making them potentially unfit for video streaming purposes. We develop *PARIMA*, a fast and efficient online viewport prediction model that uses past viewports of users along with the trajectories of prime objects as a representative of video content to predict future viewports. We claim that the head movement of a user majorly depends upon the trajectories of the prime objects in the video. We employ a pyramid-based bitrate allocation scheme and perform a comprehensive evaluation of the performance of *PARIMA*. In our evaluation, we show that *PARIMA* outperforms state-of-the-art approaches, improving the Quality of Experience by over 30% while maintaining a short response time.

## CCS CONCEPTS

• **Information systems** → **Data mining**; **Multimedia streaming**; • **Mathematics of computing** → **Time series analysis**; • **Computing methodologies** → **Supervised learning by regression**.

## KEYWORDS

360° Video Streaming, Online Learning, Adaptive Streaming

*Both authors contributed equally to this research.


*WWW '21, April 19–23, 2021, Ljubljana, Slovenia*

ACM ISBN 978-1-4503-8312-7/21/04.
https://doi.org/10.1145/3442381.3450070

**ACM Reference Format:**
Lovish Chopra, Sarthak Chakraborty, Abhijit Mondal, and Sandip Chakraborty. 2021. PARIMA: Viewport Adaptive 360-Degree Video Streaming. In *Proceedings of the Web Conference 2021 (WWW '21), April 19–23, 2021, Ljubljana, Slovenia.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3442381.3450070

## 1 INTRODUCTION

360° videos have recently captured attention in the industry [1, 2, 8] as well as in academia [13, 18, 21, 30, 32] due to their immersing and fascinating experience. Different video streaming platforms like Facebook and Youtube have introduced 360° streaming as a part of their website and applications. However, one of the biggest disadvantages of 360° video streaming is the large bandwidth requirement to provide a high-quality user experience. Since users have the flexibility to choose which part of the video they wish to see, the enriching experience comes with the cost of significant transfer of data, while only the part of the frame within the viewport of the video (*Field of View* of the video player) can be seen by the users. Based upon general calculations, roughly 80% of the bandwidth is wasted during the streaming of a 360° video, as a user rarely watches frames other than the viewports [32].

Due to large frame size, for a specific bandwidth, 360° videos are streamed at a lower quality than a regular video. Thus, there is a need for optimisations over 360° video streaming, which have emerged due to the large number of novel applications that they support [19]. The current standards for general video streaming over the web use *HTTP Adaptive Streaming* (HAS) or *Adaptive Bitrate Streaming* (ABR). During ABR, the streaming bitrates (quality) of the video frames are dynamically adapted based on the underlying network condition to ensure the best Quality of Experience (QoE) for the end users [25]. Considering this, the optimisations during a 360° video streaming can come from two fronts –

(a) predicting the user viewports in advance so that the maximum network bandwidth can be utilised to stream the viewport part of the frames at the best possible quality (or bitrate),
(b) deciding bitrates for the viewports as well as for the non-viewports in the frames dynamically on-the-fly to maximize the end-users' QoE with the maximum utilisation of the available network bandwidth.

Many 360° video streaming platforms [1, 2] use tile-based streaming methods [14, 20, 41], where the frames of a video are spatially

divided into $M \times N$ tiles and streamed as $MN$ chunks of tiles. Currently, most of the 360° platforms like Youtube and Facebook [2] use tiling-based methods for transferring the video frames. Recent researches [18, 28, 30, 32] in the field of 360° videos are focused on building models that can predict the viewport of a user to send only the part of the frame containing the viewport at higher quality and the rest of the frame at a lower quality, thus saving bandwidth. In other terms, at the same bitrate, a video can be viewed at a higher quality with viewport adaptive streaming since a higher proportion of the bitrate will be assigned to the viewport. However, many of these approaches for viewport-adaptive 360° streaming do not utilise the exclusive video contents to predict the future viewports or adjust to user preferences during streaming or are not suitable from a streaming perspective [28, 30]. Consequently, the existing methods are mostly limited to specific types of videos and fail to satisfy all the primary goals of video streaming, such as maximum video quality at the viewports, smoothness in temporal scale as well as in the spatial scale (smoothness in the quality while moving from one tile to an adjacent tile), and minimum re-buffering latency.

The user viewport depends on both (i) the content of the video and (ii) the personalised choice of the viewer, depicted by the past viewports. For example, in the case of a soccer match, the viewer may want to follow his/her favorite player or might be interested in the whole soccer field range, depending upon his/her personal choice. Similarly, for a concert video, the user might either focus only on the performer or might want to see the audience's reactions as well. Given such dynamic possibilities of having different viewports in the temporal scale for different viewers, viewport prediction in itself is a challenge. Consequently, such prediction would never result in a 100% accuracy; therefore, during the 360° video streaming, all the tiles of the frames need to be streamed, although the predicted viewport tiles may be streamed a higher quality than others. However, an abrupt quality change among the viewport tiles and the non-viewport tiles is also not desirable, as it would affect the video's spatial smoothness and thus, can affect the overall QoE. Finally, the prediction mechanism needs to be fast so that per-frame bitrates can be predicted in an online fashion during the videos' streaming. Given such multi-dimensional constraints, the overall optimisation of the 360° video streaming is indeed a complex prediction and control problem.

In this paper, we have developed a 360° viewport-adaptive video streaming platform. The viewport prediction model on the client-side is online and adjusts to user preferences dynamically based on the video content. *We claim that the viewport of a user depends upon the video contents along with the user's personal choice, based upon the movement trajectory of the prime objects in the video.* Our platform performs a one-time preprocessing of the video on the server-side to obtain the video's object trajectories. Our viewport prediction model, *PARIMA*, which is an augmented combination of *Passive Aggressive (PA) Regression* and *Auto-Regressive Integrated Moving Average* (ARIMA) times series models, utilises the set of previously observed viewports and the object trajectories for the upcoming set of frames to predict the viewports for that set of frames, and incrementally learns the weights in an online fashion, thus adapting to user preferences dynamically. The augmented combination combines the benefits of the two individual models to create a model efficient for video streaming. Based on the predictions, the client allocates bitrates to each of the tiles using a pyramid-based allocation scheme, allocating a higher proportion of bitrate to the tiles corresponding to the predicted viewport and maintaining the QoE of the user. We do not model bitrate-adaptive streaming as a part of this research and assume the user bandwidth to be constant.

We evaluate our model on two publicly available data sets, one consisting of 5 videos with head movement data for 59 users, while the other consisting of 9 videos watched by 48 users, each video having a wide range of static and moving objects. We have made our code public[1] for the research community. Using *PARIMA*, we have achieved an average QoE improvement of around 35% and 78% over two baselines and an average improvement of 117% in adaptivity over a non-adaptive bitrate allocation scheme. Our model is lightweight and exhibits a prediction latency of under 1 second for a chunk size of the same duration.

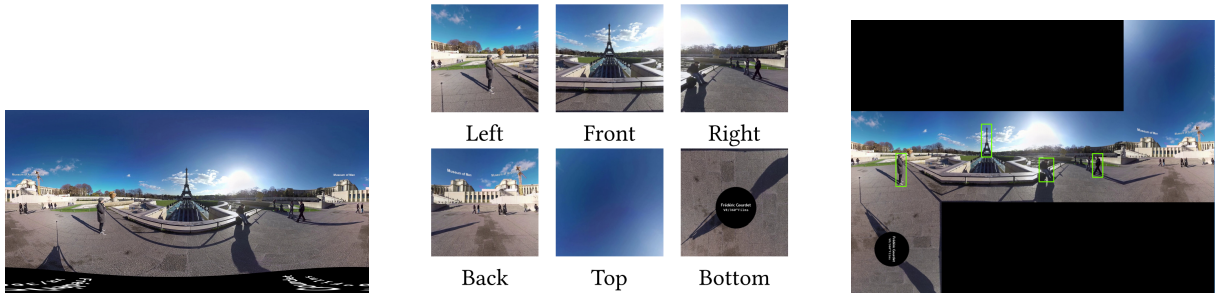
## 2 RELATED WORK

In traditional HTTP-based adaptive streaming, a video is partitioned into temporal segments, and each segment is streamed with the desired quality to minimize the bandwidth requirement while maximizing QoE for the user, thus focusing on network congestion and available bandwidth. On the other hand, optimizations in 360° video streaming involve an effort to reduce the streaming system's high bandwidth requirements by learning a model to predict the user viewports. The adaptations in 360° video streaming involve viewport-adaptive and network-adaptive streaming techniques. Viewport-adaptive streaming aims to predict the future viewport of a user by learning user head movements to allocate specific parts of the frame with a higher bitrate. In contrast, network-adaptive streaming tends to model bandwidth fluctuations to utilize network bandwidth completely. Dynamic Adaptive Streaming over HTTP (DASH) [35] is a streaming standard that adaptively streams video based on the link bandwidth between server and client.

Any video streaming platform is typically a client-server system. For 360° videos, due to the large frame size, each frame in the temporal chunks is further divided spatially into tiles [14, 20, 41], and each of these chunks of tiles is stored at different bitrates on the server-side. Each frame may typically be divided into 64–100 tiles, with each chunk of tiles being of usually 1 to 4 seconds [11]. At the back-end of the client-side, it requests the server for chunks of tiles at specific bitrates based upon the preferred quality and bandwidth available to the client. In adaptive bitrate streaming, H Mao *et. al.* [27] has proposed a reinforcement learning-based technique that learns the adaptive bitrate (ABR) algorithms adapting to a wide range of environment enhance user's quality of experience (QoE). It learns a control policy for bitrate adaptation from network throughput statistics and downloads the past few video chunks purely through experience. PARSEC [16] and SR360 [12] uses a super-resolution on the client-side to stream video under constrained bandwidth, thus reducing bandwidth requirements and improving QoE for 360° videos.

Several studies have used viewport adaptive video streaming as a part of their research. Regression-based methodologies have been studied by [33] and [10] where historical FoV trajectory is used. Works like [39] and [28] cluster users periodically based on the head

[1] https://github.com/sarthak-chakraborty/PARIMA




(a) **Equirectangular image** (b) **Cube map projection** (c) **Stitched image with sample bounding boxes of objects detected**

**Figure 1: Figure shows the equirectangular projection, different faces of corresponding cube map projection and stitched image. Image is stitched such that the cube faces share common boundary. The bounding boxes in stitched image are examples for demonstration of object tracking. In reality, many more bounding boxes will be detected in the frame.**

movement trajectory and assign new users to the existing clusters and predict the viewport. These, however, do not consider the use of video content and require an existing dataset of user viewports for any new video before predictions. Flocking based methodology is described in [36], which is applicable for a live 360° video streaming where a large number of users are available concurrently. Recent studies like *DRL360* [42] and [12] have used deep reinforcement learning-based framework to predict viewport and optimize QoE objectives across a broad set of dynamic features. However, they don't consider video content while predicting viewports.

The existing literature has studied the saliency map concept to analyze the video contents [18, 30, 31]. A saliency map shows the properties of an image at the pixel-level, where a probability map over all the tiles is used, and the bitrate is decided based on this probability distribution. Fan *et.al.* [18] in his studies developed LSTM based model that learns the sensor-related features and image saliency map to predict viewer fixation in the future. PanoSalNet [30] learns the saliency map from user viewport data using DCNN and uses the LSTM network to predict the viewport. *Mosaic* [32] makes the use of a CNN + LSTM network to find a tile probability map using a saliency map and user head movement logs as inputs. However, learning a saliency map from head movements requires a lot of training data, making the model sensitive to extending to new videos. The use of LSTM models in the above works leads to a large number of parameters, and these systems do not update the parameters throughout streaming, leading to a lack of dynamic user adaptation, making them potentially unfit for adaptive video streaming.

This work addresses the issues of the previous studies and incorporates video contents, expressly object trajectories, and past viewports of the user to predict the next set of viewports. Though saliency maps had been used as a representative for video content, those have been generated only from the existing head movement data and not using the video content explicitly. Additionally for saliency maps, multiple areas of a frame can be predicted as salient (corresponding to multiple objects in the video), and hence, the tiles outside the viewport can be given high bitrate, resulting in possibly lower QoE. Our model tends to these shortcomings, adapts to user preferences dynamically, is not heavily and statically parameterised and is easily extensible to new videos.

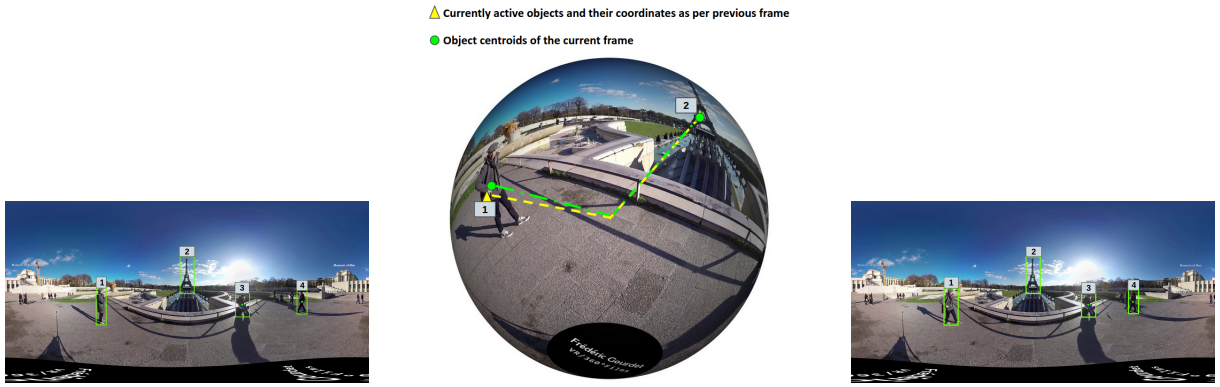
## 3 SYSTEMS OVERVIEW

Streaming 360° videos involve a collection of tasks that need to be performed in order to provide the best user experience. Our methodology involves the use of video content to predict the future viewport of the user accurately. The task of finding the contents involves a detailed analysis of the trajectories of the objects present in the video and using them efficiently for viewport prediction. The user's current viewport is detected by capturing the head movement using the movement of the device playing the video.

### 3.1 Video Preprocessing

Based upon our claim of user viewport depending upon the trajectories of prime objects in the video, we first run a one-time preprocessing of the 360° video to obtain the object trajectories on the server-side. The object trajectory meta-data required for viewport prediction can be communicated to the client before streaming. Given the input video in the form of equirectangular frames [4], we index objects over the frames such that the same objects are assigned the same indices over multiple frames. The trajectory of an object is represented by the object index and a list of coordinates of the object over various frames. Existing object trajectory algorithms [23, 40] cannot be used here since 360° videos differ from general videos in various aspects, namely, (1) objects can wrap around an equirectangular frame to emerge from another side of the frame, and (2) the equirectangular frames are distorted and typically cannot be used for object detection using standard algorithms. Although there exist a few methods for object tracking over 360-degree videos, they either (1) work for single object tracking, or (2) run object tracking in equirectangular space. Consequently, the existing techniques fail in our case because we want more efficient multi-object trajectories for viewport prediction. Hence, we develop a robust methodology that can effectively track objects in 360° videos, which we describe below.

*3.1.1 Equirectangular to Cube-Map Conversion:* 360° frames in the equirectangular form are not ideal for image processing purposes because the frames are distorted in nature (Figure 1(a)). The distortion increases as we go near the poles. Hence, we convert the equirectangular frames to their cube map projection [6], which is the least distorted version of a 360° frame as it projects the frame on




(a) **Bounding boxes from Figure 1(c) re-projected to equirectangular frame and assigned initial object IDs**

(b) **Indexing objects for next frame in spherical space**

(c) **Assigning object ids based upon the minimum spherical distance between active and next frame objects**

**Figure 2: Spherical Object tracking for 360° videos**

six sides of a cube. The conversion is carried out by first converting the equirectangular frame to its corresponding spherical projection. In the second step, points in the spherical projection are then mapped to their corresponding faces in a cube-map. An example of the conversion from equirectangular to cube-map projection is shown in Figure 1(a) and 1(b).

*3.1.2 Frame Stitching and Object Detection:* After converting the frames to their cube map projection with the distortion issue solved, we need to detect the objects present in each frame. However, in the conversion, since each pixel of an equirectangular frame is allocated a unique face of the cube, pixels of a single object might split and get mapped to different faces depending on its location on the sphere. Hence, if we run any regular object detection algorithm on each face of the cube separately, we might either not be able to detect the object or detect it as two different objects. To overcome this issue, we stitch the cube's different faces to form a single undistorted image (Figure 1(c)). The stitching ensures that any object mapped to adjacent faces of the cube gets treated as an entire entity, ensuring that object detection and tracking are continuous.

We have used YOLOv3 [34] algorithm on the stitched image of each frame to detect the objects and obtain their bounding box coordinates (Figure 1(c)), which is effective in detecting objects due to the undistorted continuous nature of the image. We explicitly eliminated object classes because we can have multiple objects of the same class in a frame. The bounding box coordinates necessarily determines the current focus of the user among multiple available objects.

After the bounding box coordinates for each object is obtained, they are then reverse-translated back to their equirectangular projection. This is because the object tracking algorithm and the viewport prediction model run in the equirectangular space.

*3.1.3 Object Tracking:* The final step in our pipeline for video preprocessing is to track the objects identified. Tracking objects essentially involves tagging them and assigning the same ID to 'close' objects in successive frames. As discussed earlier, a major difference in 360° videos is the flexibility of an object to wrap around the frame horizontally, which will be continuous in a spherical view but discontinuous in the equirectangular projection, in the case of which it should be assigned the same ID. Hence, we have devised a robust approximate spherical centroid object tracking algorithm that can index objects in 360° videos efficiently. Here is a descriptive detail of the methodology used:

(1) Compute the centroid of the equirectangular bounding boxes of objects for all frames.
(2) For the initial frame, allocate each object a unique ID. These are the *currently active* objects (Figure 2(a)).
(3) For the subsequent frames:
   (a) Project each centroid of the objects in the frame and currently active objects to spherical projection similar to Figure 2(b) by casting the latitude and longitude to their corresponding spherical coordinates.
   (b) For each pair of new frame centroids and active centroids, find the solid angle subtended by the spherical sector, which has the pair at diametrically opposite ends.
   (c) For each object $O_i$ in the current frame, find the active object $O_j^{active}$, such that:

$$solidangle(O_i^{current}, O_j^{active}) <= solidangle(O_i^{current}, O) \; \forall \; O \in O^{active}$$
$$solidangle(O_i^{current}, O_j^{active}) <= solidangle(O', O_j^{active}) \; \forall \; O' \in O^{current}$$

   where $O^{current}$ is the set of objects detected in the current frame, $O^{active}$ is the set of active objects up to the previous frame. (Note that both ways need to be satisfied). Assign the $O_i^{current}$ with the same ID as the corresponding active object $O_j^{active}$ (Figure 2(c)).
   (d) *Activate New Object*: If an object $O_i^{current}$ in the current frame has not been assigned any existing active object, it means that it has appeared for the first time in the video. In this case, we assign the object a new ID and set $O^{active} = O^{active} \cup \{O_i^{current}\}$
   (e) *Deactivate Old Objects*: If an active object $O_k^{active}$ is not assigned to any new object in the current frame, it means that either the object has disappeared or it went undetected. If the number of consecutive frames for which the

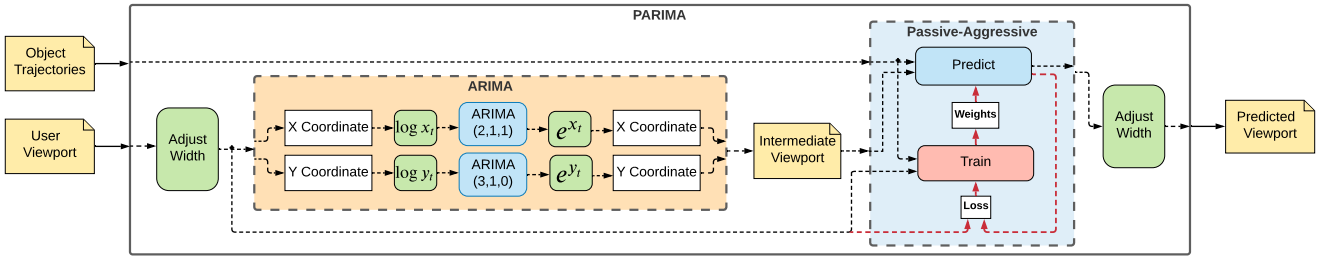



**Figure 3: PARIMA Viewport Prediction Model**

object is not assigned any new object crosses a heuristic threshold of 30, we declare the object to have disappeared, and we set $O^{active} = O^{active} - \{O_k^{active}\}$. However, if it reappears within 30 frames, it will be assigned the old object ID, the object coordinates for the missing frames are interpolated using the last available active and the new coordinates.

Thus, we will obtain a set of centroid coordinates for each object index that was detected. The algorithm is robust because it uses the solid angle between two centroids in spherical projection and hence, takes care of the object wrapping issue discussed above, as well as interpolates object coordinates for missing frames. The window of 30 frames helps to overcome the inconsistencies of the YOLO algorithm where an object might go undetected for some frames in between.

Earlier, we have argued that stitching the different faces of the cube of a cubemap projection alleviates the problem of a distorted object either being detected as two separate object entities or not being detected at all. On the contrary, as shown in Figure 1(c), the 'left' and the 'back' faces of the cube were not stitched and an object transitioning among these two faces might still suffer the same fate. However, such minor inconsistencies are countered inherently in our object tracking algorithm, where a heuristic window of 30 frames along with spherical tracking technique helps to assign same IDs to the object for the missing frames. If the object is detected as two object entities (lower probability), one of them will disappear post-transition and will not be further considered by the model(Section 3.2). The use of previous viewport in the model further alleviates the problem, and hence we get a consistent representation of the video contents.

## 3.2 Viewport Prediction

Predicting the upcoming viewport from the past viewports and the video metadata is a challenging task, especially when we want to model a dynamic system based on the videos' content. The learning task must be fast, accurate, online and should incrementally update the model weights to be able to adapt to user preferences quickly.

An important and obvious requirement of the streaming model is to be able to predict multiple frames in the future at a single instance and stream them in the form of temporal chunks of specific $t$ seconds duration, rather than just predicting a single frame at a time, in order to maintain the QoE for the user. However, a larger chunk size can compromise the prediction accuracy, since user viewport tends to vary significantly within larger chunk duration, while these changes get reflected in the model only after the chunk is streamed. We evaluate the optimal chunk size duration in Section 5.1, based upon which, we use 1 second chunk size ( = $fps$ number of frames) for viewport prediction.

We have devised a model named *PARIMA* (Figure 3), which is an augmentation of ARIMA [7] time-series model with Passive Aggressive Regression [15], to predict user viewport effectively. The set of viewports for the next chunk of frames are predicted using the previous chunk's viewports and the object coordinates for the upcoming chunk of frames, obtained from Section 3.1.

Time series models are often used in predicting head movements of users [20]. Hence, it can also be used to predict the next viewport of a user while watching a 360° video because the next viewport is essentially a temporal function of user head movement. The model takes as input the $x$ and $y$ coordinates (horizontal and vertical components respectively) for the viewports of the previous chunk of frames to predict the viewports for the next chunk of frames. Let $(x_f, y_f)$ be the viewport at frame $f$. However, if the viewport wraps around the equirectangular frame to another side of the frame, it would create a discontinuous time series of viewports. To model it as a continuous time series, $x_f$ needs to be width-adjusted, which is performed by the *'Adjust Width'* component in Figure 3 using the following transformation:

$$x_f := \begin{cases} x_f + width & \text{if } [|x_f + width - x_{f-1}| < |x_f - x_{f-1}|] \\ x_f - width & \text{if } [|x_f - width - x_{f-1}| < |x_f - x_{f-1}|] \\ x_f & \text{otherwise} \end{cases} \quad (1)$$

To overcome the case of a viewport component in the time series being negative and hamper further calculations, we shift the entire series to the right by $width$. It is to be noted that any viewport position $x_f$ is essentially same as $x_f + width$ for 360° videos because of wrapping-around property.

In order to remove inconsistencies from the data that can cause the entire viewport of the chunk from having identical values (thus generating a positive semi-definite auto-covariance matrix), we add *random(0, 0.1)* to the viewport coordinates. Augmented Dickey-Fuller test [24] showed that projecting the entire viewport data into logarithmic domain is necessary to maintain stationarity of the time series. The viewport coordinates $x_f$ and $y_f$ are then transformed to $log(x_f)$ and $log(y_f)$ respectively. These chunks of transformed x and y coordinates go into separate ARIMA models to obtain the

future chunk of viewports in logarithmic domain, which are reverse-transformed to the viewport space by simple exponentiation (see Figure 3: *'ARIMA'* block).

The above set of intermediate viewports obtained is fed as an input to the Passive-Aggressive Regression model[2]. *General Model Definition*: Passive-Aggressive Regression [15] is an efficient online learning regression algorithm that computes the mapping $f : \mathbb{R}^n \rightarrow \mathbb{R}$, $f(\mathbf{x}; \theta) = \theta^T \mathbf{x}$ where, parameters $\theta$, predictors $\mathbf{x} \in \mathbb{R}^n$. The algorithm uses the Hinge Loss Function, given by:

$$L(\theta, \epsilon) = max(0, |y - f(\mathbf{x_t}; \theta)| - \epsilon) \tag{2}$$

where $y$ is the actual value of the response variable. The parameter $\epsilon$ determines a tolerance for prediction errors. The weight update rule for PA Regression is:

$$\theta^{t+1} = \theta^t + \alpha \frac{max(0, |y_t - \theta^T \mathbf{x_t}| - \epsilon)}{||\mathbf{x_t}||^2 + \frac{1}{2C}} sign(y_t - \theta^T \mathbf{x_t}) \mathbf{x_t} \tag{3}$$

We run a coupled Passive-Aggressive Regression model that predicts the $x$ and $y$ coordinates of the viewport for the next set of frames (see Figure 3: Passive-Aggressive Block). Along with the intermediate viewport, the model uses the object trajectories that were pre-calculated in Section 3.1. The equations for the predictions for each frame in the future chunk of viewports are given by:

$$\begin{aligned} X'_f &= \theta_{0x} + \theta_X . X_f^{ARIMA} + \sum_{i=1}^{N_{obj}} \theta_{ix} . O_{Xif} \\ Y'_f &= \theta_{0y} + \theta_Y . Y_f^{ARIMA} + \sum_{i=1}^{N_{obj}} \theta_{iy} . O_{Yif} \end{aligned} \tag{4}$$

where $(X'_f, Y'_f)$ is the predicted viewport of the PARIMA model, $(X_f^{ARIMA}, Y_f^{ARIMA})$ is the intermediate viewport obtained for frame f using ARIMA model, $(O_{Xif}, O_{Yif})$ coordinates for the $i^{th}$ object for frame $f$ and $\theta$ values are the model parameters.

Since the predicted viewport $(X'_f, Y'_f)$ was initially width-adjusted using equation 1 in order to take care of wrapping around of viewports, we apply a mod $width$ on the x-coordinate of the predicted viewport to back-transform it within the equirectangular frame:

When the next chunk of frames is rendered in the video streaming, the actual set of user viewports are obtained, and the weights of the object features of the Passive Aggressive Regression model are updated using the predicted and actual values of viewports as per the update rule in Equation 3. At the start of the whole process, we train the regression model with an initial $5 * fps$ frames to prevent prediction from being 0 at the start. For these sets of frames, we can either download the whole equirectangular frame and assume the next frame's predicted view to be the same as the actual viewport of the previous frame. Figure 3, along with the above equations, shows the computation for each chunk of frames, with the same computation being repeated for all the chunks. It is to be noted that for each chunk, we create a new ARIMA model while the same Passive-Aggressive model is trained over various chunks and reused (Red arrows in Figure 3 indicates that memory is preserved and the model gets trained via this path). The viewport prediction algorithm of *PARIMA* is formulated in Algorithm 1.

[2]We have used *creme* [22], a Python library for online Machine Learning. We have modified the library code to accommodate our demands. One such change is fitting the model for $f$ frames at a single instance.

**Algorithm 1:** GET_PARIMA_VIEWPORT($V, M$)
PARIMA based Viewport Prediction

**Input:** Object Trajectories $O_f$ $\forall f$ frames, Streaming Viewport of the user $V_f$ $\forall f$ frames, PA model $M^{[1]}$
**Output:** Predicted Viewport for all frames

Initial Training of $5fps$ frames on model $M^{[1]}$;
Initialise list of Predicted Viewport Chunks $PV$;
**for** *each chunk c* **do**
  Initialise list of Predicted Viewports for Chunk $c$: $PV_c$;
  $F^{c-1} \leftarrow$ list of frames in chunk $c-1$;
  $F^c \leftarrow$ list of frames in chunk $c$;
  Initialise chunk size $cs = fps$;
  $(X_{F^{c-1}}, Y_{F^{c-1}}) \leftarrow$ horizontal and vertical components of $V_{F^{c-1}}$;
  Adjust Width of $X_{F^{c-1}}$ according to Eq. 1;
  Make series $(X_{F^{c-1}}, Y_{F^{c-1}})$ stationary;
  Initialise $M_x^{[2]} = ARIMA(2,1,1), M_y^{[2]} = ARIMA(3,1,0)$;
  Train $M_x^{[2]}$ on inputs $X_{F^{c-1}}$ to get $X_{F^c}^{ARIMA}$, $M_y^{[2]}$ on inputs $Y_{F^{c-1}}$ to get $Y_{F^c}^{ARIMA}$;
  **for** *frame* $F_f^c \in [F_1^c, F_{cs}^c]$ **do**
    $X'_{F_f^c} \leftarrow M^{[1]}(O_{XF_f^c}, X_{F_f^c}^{ARIMA})$ according to Eq. 4;
    $Y'_{F_f^c} \leftarrow M^{[1]}(O_{YF_f^c}, Y_{F_f^c}^{ARIMA})$ according to Eq. 4;
    Adjust Width of $X'_{F_f^c}$ by applying mod$width$ ;
    Append $(X'_{F_f^c}, Y'_{F_f^c})$ to $PV_c$;
  Append $PV_c$ to $PV$;
  Train $M^{[1]}$ on inputs $O_{F_1^c} \rightarrow O_{F_c^c}$ and actual viewports $V_{F_1^c} \rightarrow V_{F_c^c}$;
**return** $PV$;

ARIMA time series model helps to maintain the locality information and gives smooth predictions, while Passive-Aggressive Regression tries to update the model weights at any update step in such a way that the predicted value is as close to the actual value as possible, leading to better adaptation with fast iterations. The model update after every chunk ensures that it adapts to the user preferences dynamically. The coefficients of the object coordinates essentially represent the significance of that object, indicating user preferences.

## 3.3 Bitrate Allocation

Once we obtain the predicted set of viewports for a chunk in the form of equirectangular coordinates, we map them to the appropriate tile number since we use tiling-based streaming for the system. As the next step, tiles in each frame need to be allocated bitrates based upon the available bandwidth. 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

tiles. It is also essential to note that for a particular chunk of frames, multiple tiles might be predicted as viewports since a user might span across multiple tiles within the chunk. Hence, all the tiles corresponding to the viewport need to be given a higher bitrate than the rest. The bitrate should reduce gradually as we move away from the viewport, to maintain an optimal experience for a user.

**Algorithm 2:** SELECT_BITRATES($T_f, B_p$)
Assign Bitrates to Chunks

**Input:** Predicted Tiles for chunk c: $T_{F^c}$, Preferred total bitrate: $B_p$
**Output:** Allocated Bitrates for chunk c: $B^c$

Initialise Bitrate $B^c$;
Initialise $weight_{ij} = 1$ for each tile $(i, j)$;
Initialise chunk size $cs = fps$;
**for** *frame* $F^c_f \in [F^c_1, F^c_{cs}]$ **do**
  $(i', j') \leftarrow T_{F^c_f}$, predicted viewport tile;
  $weight_{i'j'} \leftarrow weight_{i'j'} + 1$;
  **for** *all* $(i, j) \neq (i', j')$ **do**
    $d_{ij} \leftarrow$ min. manhattan distance from $(i', j')$;
    **if** $(i, j)$ *is within Video Player FoV* **then**
      $weight_{ij} \leftarrow weight_{ij} + 1 - \frac{d_{ij}}{2 * max(d_{ij})}$;
    **else**
      $weight_{ij} \leftarrow weight_{ij} + 1 - \frac{d_{ij}}{max(d_{ij})}$;
$B^c_{ij} \leftarrow \frac{weight_{ij}}{\sum_{i,j} weight_{i,j}} B_p, \forall (i, j)$;
**return** $B_c$;

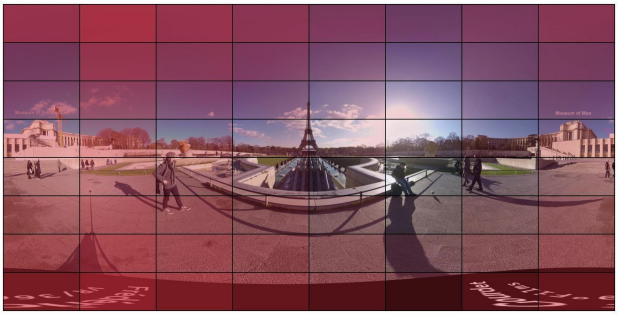


**Figure 4: General Pyramid Model of Bitrate Allocation. The figure depicts allocation for a frame with $8 \times 8$ tiling, which has viewport at tile (6, 4). Lower opacity signifies higher proportion of bitrate to be allocated to the tile**

We have incorporated *pyramid-based bitrate allocation scheme* [20] in our model (Figure 4). To assign a distribution of bitrates to the tiles $(i, j)$ in each chunk $c$, we use a weight function that would capture the proportion of 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 weight of the farthest tiles is increased by the least amount. Care is taken to allocate a higher bitrate for tiles within the Video Player FoV to maintain the Quality of Experience of the user. The distance of a tile from the viewport $(i', j')$ is measured as the minimum Manhattan distance because of the wrapping-around property of 360° frames. Due to the wrapping around property, the maximum possible distance between two tiles is $(m + n)/2$, where the tiling is $m \times n$. The weight function is then normalized, such that the net weight across all tiles is one and then used to assign bitrates to each tile proportionally as formulated in Algorithm 2.

## 4 EVALUATION TESTBED

In this section, we give a brief description of the setup, datasets and the metrics used for the evaluation of our model. We have run the object trajectory algorithm on an Intel Xeon Gold 6152 processor with 88 cores (typical desktop hardware works as well), while the results for viewport prediction, bitrate allocation, and streaming client have been generated using simple desktop having Intel i5-4210U-quad-core processor and 8GB RAM.

**Datasets:** We use two popular datasets containing several 360-degree videos of different categories along with head tracking logs. The first dataset (ds1) [13] includes five videos freely viewed by 59 users each with each video watched for 70 seconds. The second dataset (ds2) [38] has nine popular videos watched by 48 users with an average view duration of 164 seconds. Each trace of the head tracking logs for both the datasets consists of the user head position in terms of unit quaternions $(w, x, y, z)$ along with the timestamp, which is converted to equirectangular viewport using the algorithm suggested by Nguyen et. al. [29][3]. We have used the first 60 seconds of data for all the videos in our evaluation.

**Baselines:** The state-of-the-art baselines against which we have compared the performance of *PARIMA* are described below:

- *PanoSalNet:* PanoSalNet [30] learns a panoramic saliency map for each 360° frame by training a Deep ConvNet (DCNN) inspired architecture. The saliency maps are then passed along with user head movement data for the viewport prediction via an LSTM architecture. We have used the already available public code[4] as our baseline.
- *Cluster Viewport:* This method [28] clusters users based upon their viewport history. It performs predictions for a new user by finding the cluster that the user belongs to and then use quaternion extrapolation to get the next chunk of viewports. For this approach, we have implemented the model with a prediction window of 1 second and have used *pyquaternion*[5] library for quaternion-related calculations. From here on, we will refer to this model as *Clust* for convenience.
- *Non-Adaptive Bitrate Allocation (NABA) Model:* To verify the adaptivity of our model, that is, whether *PARIMA* can allocate bitrates intelligently to increase QoE, an important baseline to judge our model against is non-adaptive 360° video streaming. Under this streaming model, there is no viewport-adaptation and hence, all tiles get an equal proportion of bitrate. In general, if $B$ is the preferred bitrate of

[3]https://github.com/phananh1010/PanoSaliency (Access:September 2, 2021)
[4]https://github.com/phananh1010/PanoSalNet (Access: September 2, 2021)

streaming and the video is spatially divided into $M \times N$ tiles, then the bitrate allotted to each tile will be $B/(M \times N)$.

On one hand, while *PanoSalNet* is a supervised learning strategy that uses saliency maps (and hence, indirectly video contents) to predict viewport, *Clust* is a recent state-of-the-art algorithm, which uses unsupervised learning to cluster users and applies quaternion extrapolation for every chunk. Using the above mentioned viewport-adaptive streaming techniques having diverse methodologies yet being congruent to our study, we established that our choice of video content representation is more apt.

**Metrics:**

- *Prediction Metrics:* For evaluating the accuracy of viewport prediction of our model, we have used Manhattan Tile Error as our metric. Tile error denotes the minimum Manhattan distance between the actual tile and the predicted tile, averaged over the video length. The Manhattan Error is reported as average over all frames and over all users for the video.
- *QoE metrics:* User perceived quality is measured in a deterministic fashion using several QoE metrics that we define to empirically evaluate our model performance.

(1) The first QoE metric ($Q_1$) is the average bitrate consumed by the user in the actual viewport. In essence, it denotes the quality of the video perceived by the user. 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

$$Q_1^c = \frac{1}{n_c} \sum_{i=1}^{f_c} \left( \frac{\sum_{P_w \times P_h} a_{x,y}^i B_{x,y}^c}{tiles(P_w)_n \times tiles(P_h)_n} \right) \tag{5}$$

where, $f_c$ is the number of frames in the chunk $c$. $B_{x,y}^c$ is the allocated 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 and 0 otherwise. $tiles(P_w)_n \times tiles(P_h)_n$ is the total number of tiles in viewed in video player. $n_c$ is a normalizing constant which is calculated by the number of distinct viewport tiles in chunk $c$.

(2) The second QoE metric ($Q_2$) is a measure of the variation of the bitrate within the viewport for each frame. This metric ensures that we have minimum variation in bitrates of various tiles within the video player viewport. For chunk $c$,

$$Q_2^c = \frac{1}{n_c} \sum_{i=1}^{f_c} StdDev\{B_{x,y}^c : \; x \in tiles(P_w), \; y \in tiles(P_h)\} \tag{6}$$

(3) The third QoE ($Q_3$) captures the variation of bitrate 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 frame $f_2$. For chunk $c$,

$$Q_3^c = \frac{1}{n_c} StdDev\{ \frac{\sum_{P_w \times P_h} a_{x,y}^i B_{x,y}^c}{tiles(P_w) \times tiles(P_h)} : a_{x,y}^i = 1; \; \forall i \in f_c, \; x \in \mathcal{X}, y \in \mathcal{Y} \} \tag{7}$$

(4) The fourth QoE ($Q_4$) captures the variation of viewport bitrate across successive chunks, which is essential to minimise for good Quality of Experience. For chunk $c$,

$$Q_4^c = |Q_1^c - Q_1^{c-1}| \tag{8}$$

For all the chunks C, the aggregate QoE is given by:

$$Q = \sum_{c=1}^{C} (Q_1^c - Q_2^c - Q_3^c) - \sum_{c=2}^{C} Q_4^c \tag{9}$$

In our results, we report the QoE for videos averaged across all users.

**Hyper parameters:**

- *Tiling:* $8 \times 8$, amounting to total of 64 tiles.
- *Chunk size:* 1 second, as discussed in Section 3.2 and 5.1.
- *ARIMA Time Series Model Order (p,d,q):* $(2,1,1)$ for the x-coordinate and $(3,1,0)$ for the y-coordinate. This is achieved by hyper parameter tuning.
- *PA Regression Hyper parameters:* $C = 0.01$, $\epsilon = 0.001$. This is achieved by hyper parameter tuning.
- *Video Player Dimension:* $600 \times 300$
- *Preferred Video Bitrate by User:* Assumed to be constant at 8 Mbps (1080p) for the experiment.

## 5 EVALUATION RESULTS

This section discusses the various experiments that we have performed to demonstrate the effectiveness of *PARIMA*. We have compared our model against the baselines mentioned in Section 4, as well as separately perform an individual assessment of the viewport prediction method. The results for the given datasets are generated using an emulator, which replicates the model to behave the same way as it would in a general 360° setup. However, we also developed a basic video streaming system, where we have used a fast Kvazaar HEVC [3, 37, 41] encoder-decoder along with GPAC MP4Box [26] for the creation of HTTP DASH segments, which ensures that the decoding time is low and is not a bottleneck in streaming. We use MP4Client [26] for streaming client development.

### 5.1 Optimal Chunk Size Prediction

This subsection discusses the choice of the optimal chunk size/prediction window for the model. Choosing an optimal chunk size essentially brings forth a trade-off between the streaming time and the prediction accuracy. An underlying trade-off also occurs where a client may need to store a considerable amount of video for a smooth experience, while prediction algorithms limit the amount of data to be stored in buffer [9]. A smaller chunk size facilitates better prediction accuracy and QoE for the user because of more frequent model updates. In contrast, a larger chunk size would suffer from a poor prediction model with low QoE. On the other hand, streaming time would have to be low for the chunks to ensure smooth streaming of video without buffering. We analysed *PARIMA* for four different chunk duration: 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).

To justify our claim of shorter chunks producing better QoE, we run the model on the five videos in dataset ds1 for the various chunk sizes and report the QoE for each video, averaged across all chunks for all users. As expected based upon the above argument, Figure 5 shows a strict decrease in the average QoE for all the videos as we

increase the chunk size. Similar results were obtained for dataset ds2. Hence, the smaller chunk size is always preferable over larger ones in terms of QoE.

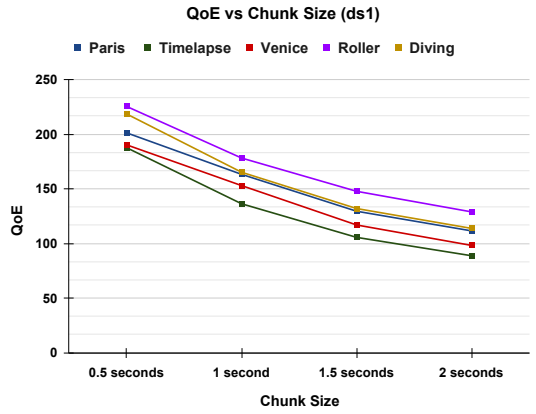



**Figure 5: *QoE of videos for varying chunk sizes***

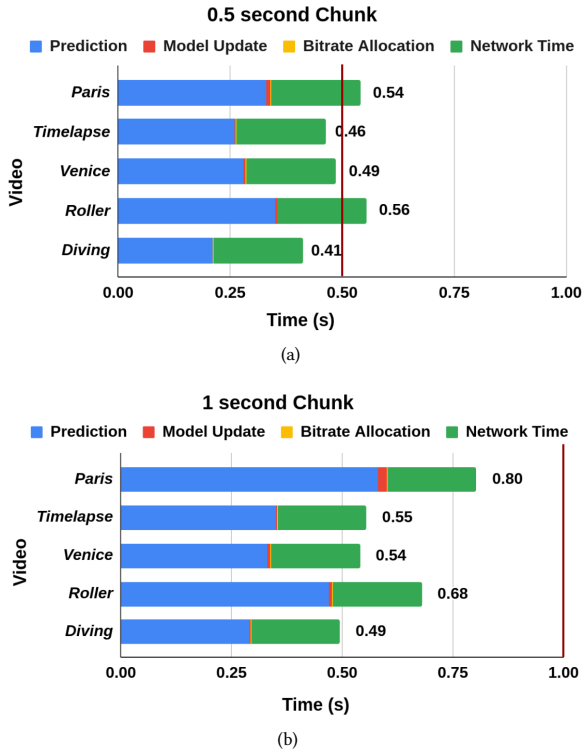



**Figure 6: *Comparison of streaming times of chunks of duration 0.5 seconds and 1 second***

However, although a chunk size of 0.5 seconds has a better QoE than a chunk of 1 second, it is not efficient from a video streaming point of view regarding the total streaming time of a chunk in a video. To argue over this fact, we find the average total streaming time for a chunk for the videos in ds1 averaged over all chunks for all users for chunk sizes of 0.5s and 1s (Figure 6). The streaming time is essentially composed of four parts: (1) model update time, (2) viewport prediction time, (3) bitrate allocation time, and (4) network time. The network time includes the time taken by the client to send a request and receive the video chunk from the server, decode the chunks, and stream them. As evident from the graphs, the model update and bitrate allocation time is meagre and not a bottleneck for any of the videos, whereas the prediction time is comparatively high. To facilitate our model even on 3G networks, we have considered the network latency to be 150 ms [17]. We use the video-streaming system developed to find the average decoding and streaming time of the chunks to be 50ms, making a total network time of 200ms for one chunk. We observe that for the chunk duration of 0.5s, the total streaming time is very close, and in some cases, exceeding 0.5 seconds. On the other hand, the total streaming time is comfortably under 1 second for a chunk duration of 1s, thus facilitating smooth video streaming. Also, MP4Box [11, 26] typically generates DASH segments of 1 second, also supported by Flare [33], which further emphasizes on using 1 second over 0.5 seconds as our chunk size. Similar results were obtained on ds2. If a chunk of a certain time duration $t$ seconds takes more time than $t$ seconds to stream the chunk, it would essentially mean that the video will buffer at every chunk, which is not desired for a video-streaming system. Hence, our claim of using a chunk size of $fps$ (chunk duration 1 second) is justified.

## 5.2 Enhancement over Individual Models

We claimed that *PARIMA* combines the benefits of the Passive-Aggressive Regression and ARIMA time series model. Thus, in order to justify our claim, we run viewport prediction for Passive-Aggressive Regression and ARIMA time series models separately and compare the results with *PARIMA* model.

For the Passive-Aggressive Regression model, the predicted viewport for a specific frame is computed using the objects' coordinates in that frame and the predicted viewport of the previous frame. For the ARIMA model, on the other hand, the predicted viewport for a chunk of frames is simply obtained using the actual viewports of the previous chunk, which is essentially the same as the 'Intermediate Viewport' in Figure 3. We report QoE and Manhattan tile error for the three models for different videos averaged over all users. We plot the QoE for each model on dataset ds1 in Figure 7 and tabulate the Manhattan Tile Error in Table 1.

| Model | Paris | Diving | Roller | Timelapse | Venice |
|---|---|---|---|---|---|
| **PARIMA** | 0.612 | 0.337 | 0.234 | 0.685 | 0.353 |
| **PA** | 1.366 | 1.374 | 1.412 | 1.097 | 1.130 |
| **ARIMA** | 0.643 | 0.368 | 0.305 | 0.779 | 0.438 |

**Table 1: Average Manhattan Tile Error for PARIMA, PA Regression, ARIMA Time Series Models**

We can see that *PARIMA* has a superior QoE and a lower tile error than the individual models. Passive-Aggressive Regression suffers from high tile errors due to the propagation of error over multiple frames. Since the predicted output of one frame is fed as

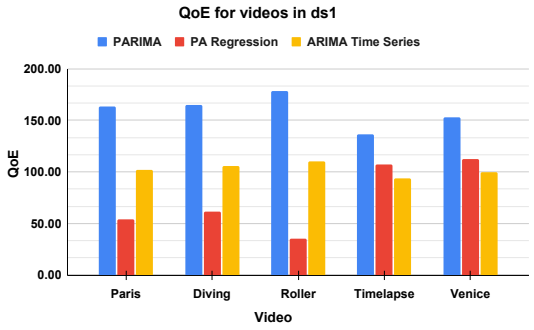
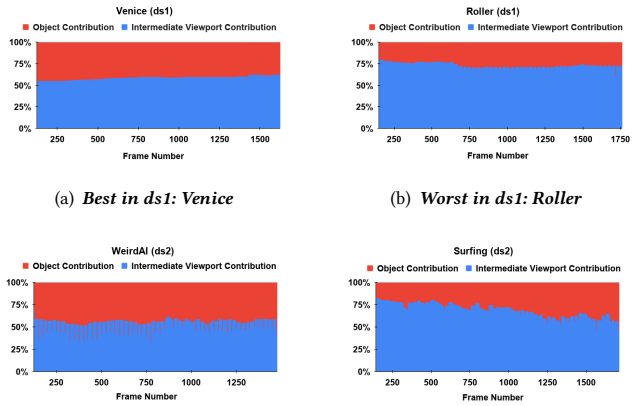



**Figure 7: *QoE of PARIMA, PA Regression and ARIMA Time Series Models***




(a) ***Best in ds1: Venice*** (b) ***Worst in ds1: Roller***

(c) ***Best in ds2: WeirdAI*** (d) ***Worst in ds2: Surfing***

**Figure 8: Percentage contribution of objects and past viewport in final prediction from *PARIMA*. Plots show the object contribution over the videos' length having the best and worst average object contribution %.**

an input to the next frame, the error in a certain viewport prediction gets propagated to the further frames. The Passive-Aggressive component shows great adaptivity due to its online nature and efficiently uses object trajectories to determine significance of various parts of the frames, leading to an overall useful model for viewport prediction. Similar inferences were obtained for ds2. ARIMA time series model, on the other hand, has low prediction errors and strengthens the Passive-Aggressive component when coupled into the *PARIMA* model. The combined model has higher QoE and lower prediction errors than any of the two models, which validates the claim that the augmentation of the Passive Aggressive Regression and ARIMA models generates a superior model.

## 5.3 Measure of Object Contribution

In our research, we claimed that a user's viewport depends upon the trajectory of the prime objects in the video. Since it also remains in the vicinity of the previous viewport, we use the last information viewport in the model effectively to predict the next chunk of viewports. To verify the above claim, we evaluate the percentage contribution of object trajectories in predicting the viewport. We present the results for the X-coordinate of the viewport since it typically has higher variability compared to the Y-coordinate. The object contribution essentially determines how significant they are in predicting the viewport of the user.

From Eq. 4, we get the contribution of object trajectories in the prediction of the X coordinate of the viewport at frame $f$ as $\sum_{i=1}^{N_{obj}} \theta_{ix}.O_{Xif}$, while $\theta_{0x}+\theta_X.X_f^{ARIMA}$ is denoted as 'Intermediate Viewport Contribution'. We find the proportion of contribution of object trajectory over the length of the video, averaged across all users which is shown in Figure 8 for the best and worst average object contribution for videos in ds1 and ds2. For all the videos in ds1 and ds2, we further report the average percentage contribution across all users for all frames in Table 2.

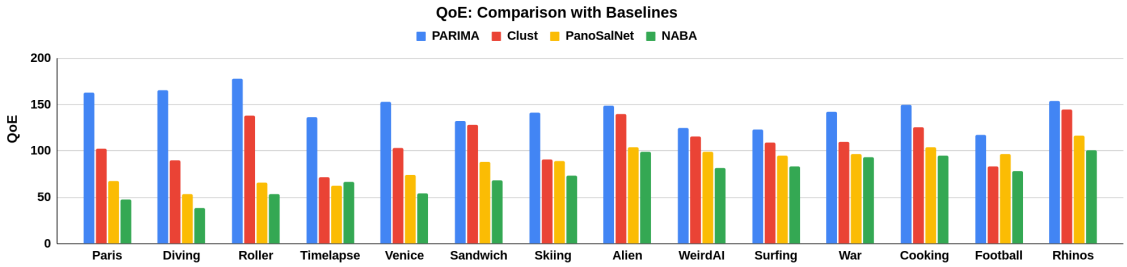
Table 2 tabulates the number of objects detected by our object tracking algorithm and the average percentage contribution of object trajectories in predicting the user viewport. As observed, the percentage object contribution varies from 27.2% to 43.6% in our model for ds2 while ranging from 27.5% to 40.1% for ds1, with an average of 33.8%. However, the standard deviation is only 5.58%,

| | Video | # Objects Detected | Avg. Object Contribution (%) |
|---|---|---|---|
| ds1 | Paris | 8 | 31.5 |
| | Diving | 41 | 30.9 |
| | Roller | 66 | 27.5 |
| | Timelapse | 61 | 39.5 |
| | Venice | 14 | 40.1 |
| ds2 | Sandwich | 41 | 27.4 |
| | Skiing | 77 | 31.5 |
| | Alien | 92 | 32.1 |
| | WeirdAI | 25 | 43.6 |
| | Surfing | 40 | 27.2 |
| | War | 397 | 34.5 |
| | Cooking | 1105 | 36.3 |
| | Football | 166 | 42.8 |
| | Rhinos | 77 | 27.9 |
| | **Average** | **157** | **33.8** |

**Table 2: Average Percentage Object Trajectory Contribution**

and hence object trajectories contribute to roughly one-third of the viewport prediction on average. Even though *Surfing* of ds2 shows the worst object contribution among all the other videos, it is not insignificant compared to the average of 33.8%. From Figure 8, we see that the object contribution of *Surfing* increases as we play the video further, showing that the model learns that the viewport depends on the object trajectory. For *WeirdAI*, which has the best average object contribution, a consistent value of around 40% is maintained, similar to *Venice*.

Using the above results, it is evident that object trajectories play an important role in determining the viewport of the object, as they have significant contributions in the prediction. The use of object trajectories improving the PARIMA model over the ARIMA model in Table 1 depicts that tile error reduces upon the inclusion of object trajectories, thus further justifying the claim that video




| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PARIMA | 163.29 | 165.22 | 178.23 | 136.36 | 153.01 | 132.57 | 141.75 | 149.13 | 125.11 | 123.18 | 142.15 | 149.81 | 117.56 | 153.62 |
| Clust | 102.28 | 89.59 | 137.81 | 71.47 | 103.27 | 128.43 | 90.44 | 139.71 | 115.92 | 109.11 | 109.92 | 125.78 | 83.51 | 144.55 |
| PanoSalNet | 67.79 | 53.17 | 65.71 | 62.24 | 74.50 | 87.89 | 88.80 | 104.28 | 99.36 | 94.76 | 96.62 | 103.92 | 96.49 | 116.77 |
| NABA | 47.41 | 38.34 | 53.15 | 66.77 | 53.91 | 68.44 | 73.26 | 98.69 | 82.00 | 83.22 | 93.00 | 95.30 | 78.33 | 100.68 |

**Figure 9: *QoE comparison of PARIMA with PanoSalNet, Clust and NABA models***

content, in terms of object trajectories, play an essential role in viewport prediction.

## 5.4 Baseline Comparison

Finally, to elicit the efficacy of our model, we evaluate *PARIMA* against three state-of-the-art baselines: *PanoSalNet*, *Clust* and *NABA*. We discussed in Section 4 how these baselines are congruent or effective in elucidating the effeciency of the PARIMA model.

*PanoSalNet* and *Clust* provide us with two viewport prediction models. We use them to predict viewports along with the bitrate allocation scheme discussed in Section 3.3. We find the Manhattan Tile Error and QoE for the videos in ds1 and ds2 averaged across all users across all chunks. The results for the Tile Error and QoE are shown in Table 3 and Figure 9 respectively.

| | Video | PARIMA | PanoSalNet | Clust |
|---|---|---|---|---|
| ds1 | Paris | 0.612 | 1.832 | 1.068 |
| | Diving | 0.337 | 1.934 | 1.072 |
| | Roller | 0.234 | 1.561 | 0.865 |
| | Timelapse | 0.685 | 1.874 | 1.205 |
| | Venice | 0.353 | 1.867 | 1.093 |
| ds2 | Sandwich | 0.144 | 1.645 | 1.097 |
| | Skiing | 0.156 | 2.691 | 1.337 |
| | Alien | 0.149 | 2.347 | 0.982 |
| | WeirdAI | 0.133 | 2.249 | 1.047 |
| | Surfing | 0.177 | 1.629 | 1.335 |
| | War | 0.178 | 1.661 | 1.089 |
| | Cooking | 0.159 | 1.355 | 1.484 |
| | Football | 0.175 | 2.511 | 1.130 |
| | Rhinos | 0.104 | 2.420 | 0.882 |

**Table 3: Tile Errors for PARIMA, PanoSalNet and Clust**

As it can be observed from Figure 9, *PARIMA* exhibits a higher QoE than the two baselines for all the videos, with *PanoSalNet* being the worse of the three. *Clust* does not consider video contents while predicting viewport which, as discussed in Section 5.3, play a significant role in *PARIMA*. *Clust* clusters the users based on viewport history and matches a new user to a cluster to predict the viewport. However, such a scheme can work for videos that particularly focuses on a character, as is the case for many videos in ds2. In ds1, however, the objects are spaced throughout the frame, and different users can have different viewing patterns. Hence, it becomes difficult to cluster users, or the new users can exhibit a different viewing pattern. It can be observed from Figure 9, that for videos in ds1, *Clust* performs much poorer than *PARIMA* while the two methods are comparable for many videos in ds2.

Since *PanoSalNet* uses a saliency map to capture the video's content and employs LSTM based learning, its weights are not dynamically changed w.r.t. user preferences. Thus is often fails to categorize abrupt user behavior and allocates poor bitrate distribution among the tiles. This gets translated into a poorer prediction and hence a lower QoE. *PARIMA* obtained an average improvement of 78.67% in QoE over *PanoSalNet* and 35.38% over *Clust*. In Table 3, we see similar results with *PARIMA* performing better than the other two baselines.

*Clust* can work well only when we have an existing head movement dataset of a sufficient set of users for efficient clustering, which would also essentially mean storing the head movements of all users. Similarly *PanoSalNet* use existing head movement records to generate the saliency map. *PARIMA*, on the other hand, just requires the object trajectory information, calculated one-time on the server side, and decouples the user viewport prediction from the viewports of other users, leading to lower memory/storage consumption, faster prediction and easy extensiblity to new videos. The model updates ensure that higher weights are given to the objects which the user is inclined to watch, leading to better user adaptation and hence, better QoE.

NABA model of bitrate allocation assumes no viewport prediction, and bitrate is allocated to each of the tiles in the chunk of frames equally. Hence, it is non-adaptive. As evident from Figure 9, NABA exhibits the least QoE compared to the other models. *PARIMA* performs comfortably better than NABA for all the videos with an average of 117.88% improvement in adaptivity, hence verifying viewport-adaptivity for our model.

## 6 CONCLUSION AND FUTURE WORK

In this paper, we presented a novel tile-based viewport prediction model *PARIMA*, that takes into account the video contents along with user head movement history to predict viewport for the future frames in the video. We have used object trajectories as a representative measure for the video content since they are exclusive to the video. We have shown through our experiments that while predicting the viewport, *PARIMA* assigns around 34% weightage to the object's position on an average. This verifies our claim that the viewport of a user depends not only on the previous viewport but also upon the trajectory of prime objects present in the video.

Our system uses a 1-second chunk duration for viewport prediction and streaming. We have used a pyramid bitrate allocation scheme to allocate a higher bitrate to the predicted viewport and gradually decrease it as the tiles move away from the viewport. We evaluated *PARIMA* and compared its performance against the current state-of-the-art video streaming solutions. Our evaluations show that *PARIMA* offers better QoE relative to other state-of-the-art methods.

In future, we plan to extend our work by predicting network inconsistencies and coupling the viewport-adaptive streaming methodology with network adaptivity. We also plan to incorporate audio channel as a supplemental representation of the video content, which includes various challenges including complex representation of the video and the presence of 3D audio. We plan to validate our work further by using more extensive datasets.

## REFERENCES


[1] 2017. Bringing pixels front and center in VR video. https://blog.google/products/google-ar-vr/bringing-pixels-front-and-center-vr-video/. (2017).

[2] 2017. Facebook End-to-end optimizations for dynamic streaming. https://engineering.fb.com/video-engineering/end-to-end-optimizations-for-dynamic-streaming/. (2017).

[3] 2018. HEVC(H.265): What is it and Why Should You Care? https://blog.frame.io/2018/09/24/hevc-format-wars/. (2018).

[4] 2020. Equirectangular Projection. https://en.wikipedia.org/wiki/Equirectangular_projection. (2020).

[5] 2020. Pyquaternion, Python module for representing and using quaternions. http://kieranwynn.github.io/pyquaternion/. (2020).

[6] 2020. vrProjector. https://github.com/bhautikj/vrProjector. (2020).

[7] Ratnadip Adhikari and R. K. Agrawal. 2013. An Introductory Study on Time Series Modeling and Forecasting. arXiv:cs.LG/1302.6613

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

[9] Mathias Almquist, Viktor Almquist, Vengatanathan Krishnamoorthi, Niklas Carlsson, and Derek Eager. 2018. The prefetch aggressiveness tradeoff in 360 video streaming. In *Proceedings of the ACM MMSys 2018*.

[10] Yanan Bao, Huasen Wu, Tianxiao Zhang, Albara Ah Ramli, and Xin Liu. 2016. Shooting a moving target: Motion-prediction-based transmission for 360-degree videos. In *2016 IEEE International Conference on Big Data (Big Data)*. IEEE, 1161–1170.

[11] N. Bouzakaria, C. Concolato, and J. Le Feuvre. 2014. Overhead and performance of low latency live streaming using MPEG-DASH. In *Proceedings of the IISA 2014*.

[12] Jiawen Chen, Miao Hu, Zhenxiao Luo, Zelong Wang, and Di Wu. 2020. SR360: boosting 360-degree video streaming with super-resolution. In *Proceedings of the 30th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*. 1–6.

[13] Xavier Corbillon, Francesca De Simone, and Gwendal Simon. 2017. 360-degree video head movement dataset. In *Proceedings of the ACM MMSys 2017*.

[14] Xavier Corbillon, Gwendal Simon, Alisa Devlic, and Jacob Chakareski. 2017. Viewport-adaptive navigable 360-degree video delivery. In *Proceedings of the IEEE ICC 2017*.

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

[16] M. Dasari, A. Bhattacharya, S. Vargas, P. Sahu, A. Balasubramanian, and S. R. Das. 2020. Streaming 360-Degree Videos Using Super-Resolution. In *Proceedings of the IEEE INFOCOM 2020*.

[17] Wei Dong, Zihui Ge, and Seungjoon Lee. 2011. 3G Meets the Internet: Understanding the Performance of Hierarchical Routing in 3G Networks. In *Proceedings of the ITC 2011*.

[18] Ching-Ling Fan, Jean Lee, Wen-Chih Lo, Chun-Ying Huang, Kuan-Ta Chen, and Cheng-Hsin Hsu. 2017. Fixation prediction for 360 video streaming in head-mounted virtual reality. In *Proceedings of the ACM NOSSDAV 2017*.

[19] Ching-Ling Fan, Wen-Chih Lo, Yu-Tung Pai, and Cheng-Hsin Hsu. 2019. A Survey on 360° Video Streaming: Acquisition, Transmission, and Display. *ACM Computing Surveys (CSUR)* 52, 4 (2019), 71.

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

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

[22] Max Halford, Geoffrey Bolmier, Raphael Sourty, Robin Vaysse, and Adil Zouitine. 2019. *creme, a Python library for online machine learning*. https://github.com/MaxHalford/creme

[23] Mei Han, Wei Xu, Hai Tao, and Yihong Gong. 2004. An algorithm for multiple object trajectory tracking. In *Proceedings of the IEEE CVPR 2004*.

[24] faculty.smu.edu. [n.d.]. Augmented Dickey-Fuller Unit Root Tests. ([n. d.]).

[25] Jonathan Kua, Grenville Armitage, and Philip Branch. 2017. A survey of rate adaptation techniques for dynamic adaptive streaming over HTTP. *IEEE Communications Surveys & Tutorials* 19, 3 (2017), 1842–1866.

[26] Jean Le Feuvre, Cyril Concolato, and Jean-Claude Moissinac. 2007. GPAC: Open Source Multimedia Framework. In *Proceedings of the 15th ACM International Conference on Multimedia* (Augsburg, Germany) *(MM '07)*. Association for Computing Machinery, New York, NY, USA, 1009–1012. https://doi.org/10.1145/1291233.1291452

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

[28] Afshin Taghavi Nasrabadi, Aliehsan Samiei, and Ravi Prakash. 2020. Viewport Prediction for 360° Videos: A Clustering Approach. In *Proceedings of the ACM NOSSDAV 2020*.

[29] Anh Nguyen and Zhisheng Yan. 2019. A saliency dataset for 360-degree videos. In *Proceedings of the ACM MMSys 2019*.

[30] Anh Nguyen, Zhisheng Yan, and Klara Nahrstedt. 2018. Your attention is unique: Detecting 360-degree video saliency in head-mounted display for head movement prediction. In *Proceedings of the ACMMM 2018*.

[31] Tam V Nguyen, Mengdi Xu, Guangyu Gao, Mohan Kankanhalli, Qi Tian, and Shuicheng Yan. 2013. Static saliency vs. dynamic saliency: a comparative study. In *Proceedings of the ACMMM 2013*.

[32] Sohee Park, Arani Bhattacharya, Zhibo Yang, Mallesham Dasari, Samir R Das, and Dimitris Samaras. 2019. Advancing User Quality of Experience in 360-degree Video Streaming. In *Proceedings of the IFIP Networking 2019*.

[33] Feng Qian, Bo Han, Qingyang Xiao, and Vijay Gopalakrishnan. 2018. Flare: Practical viewport-adaptive 360-degree video streaming for mobile devices. In *Proceedings of the ACM MobiCom 2018*.

[34] Joseph Redmon and Ali Farhadi. 2018. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767* (2018).

[35] Thomas Stockhammer. 2011. Dynamic adaptive streaming over HTTP–: standards and design principles. In *Proceedings of the ACM MMSys 2011*.

[36] Liyang Sun, Yixiang Mao, Tongyu Zong, Yong Liu, and Yao Wang. 2020. Flocking-based live streaming of 360-degree video. In *Proceedings of the 11th ACM Multimedia Systems Conference*. 26–37.

[37] Marko Viitanen, Ari Koivula, Ari Lemmetti, Arttu Ylä-Outinen, Jarno Vanne, and Timo D. Hämäläinen. 2016. Kvazaar: Open-Source HEVC/H.265 Encoder. In *Proceedings of the ACMMM 2016*.

[38] Chenglei Wu, Zhihao Tan, Zhi Wang, and Shiqiang Yang. 2017. A dataset for exploring user behaviors in VR spherical video streaming. In *Proceedings of the ACM MMSys 2017*.

[39] Lan Xie, Xinggong Zhang, and Zongming Guo. 2018. Cls: A cross-user learning based system for improving qoe in 360-degree video adaptive streaming. In *Proceedins of the ACMMM 2018*.

[40] Alper Yilmaz, Omar Javed, and Mubarak Shah. 2006. Object tracking: A survey. *Acm computing surveys (CSUR)* 38, 4 (2006), 13–es.

[41] Alireza Zare, Alireza Aminlou, Miska M Hannuksela, and Moncef Gabbouj. 2016. HEVC-compliant tile-based streaming of panoramic video for virtual reality applications. In *Proceedings of the ACMMM 2016*.

[42] Yuanxing Zhang, Pengyu Zhao, Kaigui Bian, Yunxin Liu, Lingyang Song, and Xiaoming Li. 2019. DRL360: 360-degree Video Streaming with Deep Reinforcement Learning. In *Proceedings of the IEEE INFOCOM 2019*.