LiveObj: Object Semantics-based Viewport Prediction for Live Mobile Virtual Reality Streaming

Xianglong Feng, *Student Member, IEEE*, Zeyang Bao, *Student Member, IEEE*, and Sheng Wei, *Member, IEEE*

Abstract— Virtual reality (VR) video streaming (a.k.a., 360-degree video streaming) has been gaining popularity recently as a new form of multimedia providing the users with immersive viewing experience. However, the high volume of data for the 360-degree video frames creates significant bandwidth challenges. Research efforts have been made to reduce the bandwidth consumption by predicting and selectively streaming the user's viewports. However, the existing approaches require historical user or video data and cannot be applied to live streaming, the most attractive VR streaming scenario. We develop a live viewport prediction mechanism, namely *LiveObj*, by detecting the objects in the video based on their semantics. The detected objects are then tracked to infer the user's viewport in real time by employing a reinforcement learning algorithm. Our evaluations based on 48 users watching 10 VR videos demonstrate high prediction accuracy and significant bandwidth savings obtained by *LiveObj*. Also, *LiveObj* achieves real-time performance with low processing delays, meeting the requirement of live VR streaming.

1 Introduction

Mobile head mounted display (HMD) devices for virtual reality (VR) applications have become increasingly popular recently. Major industry vendors, such as Facebook, HTC, and Microsoft, have released a variety of VR/AR headsets [1, 4, 5]. Thanks to the growing trend in adopting these VR headsets in the consumer market, the immersive VR videos have started to gain traction in many mobile video streaming scenarios. For example, many sports events have been broadcast via VR videos [2, 6]. YouTube has a huge number of 360-degree videos spanning documentary, sports, and performance shows [7]. In addition, news media, such as CNN, have also entered the VR era broadcasting news stories in the form of VR streaming [3]. VR streaming provides the users with a unique immersive viewing experience, in which one can freely select the intended viewport out of the 360-degree frames by conducting natural head movements, similar to what one would do in the physical world.

Despite the immersive experience provided to the end users, the wider deployment of VR videos still faces significant technical challenges, among which the escalated bandwidth consumption is the primary cause of concern. For example, a typical 720p (1280×720) VR video would be approximately equivalent to a traditional 2D video with 3840×1920 resolution. Considering the H.264 compression and 30 FPS frame rate, the 720p 360-degree video would consume around 45 Mbps of bandwidth. This would further translate to 500 Mbps of bandwidth if the video resolution is 4K (4096×2160), i.e., the state-of-the-art high definition video that end users would expect to have in an immersive experience, which is hardly achievable in household networks. The huge bandwidth consumption would overwhelm the capacity of the mobile VR headset and result in intolerable video buffering delays compromising the user's experience.

Furthermore, the bandwidth/delay issues would be significantly worsened in the highly desirable scenario of live VR streaming, which is the most attractive VR use case for live events such as sports games and breaking news. The reason behind this deficiency lies in the fact that live streaming poses strict real-time requirement on top of the already challenging bandwidth/delay issues in VR streaming. In particular, one must address two major challenges to achieve a viable live VR streaming solution: (1) the *bandwidth challenge*: An effective bandwidth optimization mechanism is required to fit the bandwidth requirement of VR streaming to the mobile network capacity; and (2) the *live challenge*: The bandwidth optimization mechanism must be real-time and compatible with live streaming, where the video content is not available in advance but generated on the fly with the live event.

To address the *bandwidth challenge*, the state-of-the-art VR streaming research has mainly focused on viewport prediction [8, 9, 12, 17, 21, 22, 29–31], which aims to save the bandwidth consumption in VR streaming by predicting the user's viewport of interest and only streaming the portion of the video that is likely to be watched by the user with high quality. Since the user can only watch one viewport at a single point of time, an accurate viewport prediction could technically reduce the bandwidth consumption down to the level of traditional non-VR video streaming. However, most of the existing viewport prediction methods, such as *video content-based* [8, 12, 17, 29, 30] and *user trajectory-based* [9, 21, 22, 31], either rely on historical user or video data or hard to achieve sufficient prediction accuracy. Therefore, they cannot meet the aforementioned *bandwidth challenge* and *live challenge* to support live VR streaming.

To effectively address both the *bandwidth challenge* and the *live challenge* in live VR streaming, we propose a user/video hybrid viewport prediction approach that employs both real-time video content and user feedback in achieving accurate and live compliant viewport prediction. Our exploration of such a viewport prediction mechanism begins with a comprehensive analysis (in Section 4) on the user viewing behavior drawn from a public user head movement dataset [27], involving 48 real world users watching 10 representative 360-degree videos. Our analysis reveals that the user's viewports of interest follow traceable patterns that are correlated with the semantics of the objects presented in the video, instead of the physical locations of the viewports.

Based on the comprehensive analysis and observation, we develop an object semantics-based live viewport prediction framework, namely *LiveObj*, which tracks the identifiable and meaningful objects in the 360-degree video and converts the object semantics into well-inferred user viewing behavior. In particular, *LiveObj* involves two major technical components: (1) an object tracker, which detects and tracks all the objects that may be of the user's interest by employing real-time object detection; and (2) a viewport predictor, which combines the object semantics and the real-time user feedback into the viewport prediction results using reinforcement learning. We evaluate *LiveObj* using a public dataset [27], which justifies its superior performance.

2 Background: Live VR Streaming

Figure 1 illustrates the overall workflow of a typical live VR streaming system. The workflow begins with a 360-degree camera capturing the 360-degree panoramic scene. The collected 360-degree frames are then partitioned into small segments by the video packager and delivered to the content distribution network (CDN) for broader distributions to the

- *Xianglong Feng, Rutgers University. E-mail: xianglong.feng@rutgers.edu.*
- *Zeyang Bao, Rutgers University. E-mail: zb95@scarletmail.rutgers.edu*
- *Sheng Wei, Rutgers University. E-mail: sheng.wei@rutgers.edu.*

Manuscript received 9 Sept. 2020; revised 15 Dec. 2020; accepted 8 Jan. 2021. Date of publication 1 Apr. 2021; date of current version 7 Apr. 2021. Digital Object Identifier no. 10.1109/TVCG.2021.3067686

clients. On the client side, the user wears the HMD to request the video segments and display the corresponding viewports determined by the head movement information obtained from the sensors on the HMD.

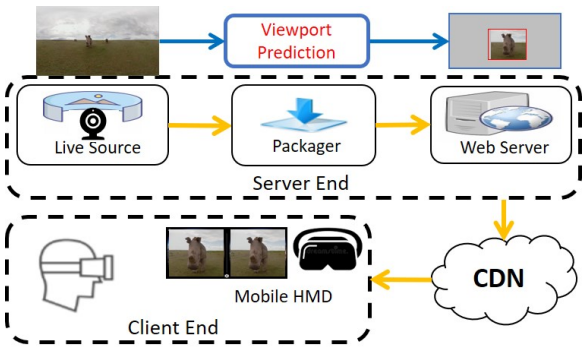


Fig. 1. Overall workflow of live VR streaming.

The viewport prediction module can be deployed at the server end as part of the live packager, as shown in Figure 1, where the video segments are generated corresponding to the predicted viewports and delivered to the CDN for content distribution. In particular, the portion of the video within the predicted viewport can be encoded with high bitrate/resolution to ensure premium quality, and the portion outside the predicted viewport can be delivered with low quality to reduce bandwidth consumption. In the worst case scenario where the viewport prediction is incorrect, the user can still watch the low quality version of the video to avoid interruptions in the video streaming experience.

3 Related Work

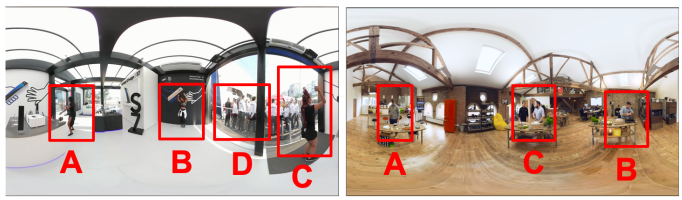
There are two categories of viewport prediction methods that have been developed. First, *video content-based* approaches focus on statistical analysis on the VR video and the user viewing history. For example, several approaches collect a group of users' viewport information watching a video and generate a heat-map to predict another group of users' viewports watching the same video [8, 17]. Other methods employ machine learning-based techniques to model the relationship between the video content and the user viewing behavior [12, 29, 30]. Second, *user trajectory-based* approaches predict the future viewport by either modeling the user's head movement trajectory in the current video session [9, 22, 31] or by clustering the trajectories of other users who have watched the target video in the past [21].

Although the existing solutions are effective in their specific scenarios, they are not yet sufficient to address either the *bandwidth challenge* or the *live challenge* in bringing 360-degree video to the front of live streaming and meeting the real-time requirement. First, the existing *video content-based* methods cannot be applied to the *live challenge*, as they rely on historical data related to the users past viewing behavior of the target video, which does not exist for live streamed videos. Second, the existing *user trajectory-based* methods have the potential to be adopted in live streaming, as they only require real-time data from the current video session. However, it is hard to achieve an acceptable prediction accuracy by leveraging only the user trajectory, especially for a long time span of a few seconds (i.e., the video buffer length), as it is highly likely that the users would change their head movement patterns during this time. As a result, the existing *user trajectory-based* methods cannot address the *bandwidth challenge* due to the potential retransmissions caused by erroneous predictions.

To date, there are only two viewport prediction methods that attempted to address both challenges to support live VR streaming, namely the motion-based method (i.e., *Motion*) [15] and the online deep learning-based method (i.e., *LiveDeep*). The *Motion* approach leverages motion tracking and dynamic user interest modeling to predict the user viewport. Thanks to the real-time performance of Gaussian Mixture Model (GMM) [34] and the feature mapping method, *Motion* could predict the user viewport by analyzing the video content and the user feedback while meeting the real-time requirement in live streaming. However, it is limited by the motion detection technique that only applies to the videos with static background. For the videos with complicated background or captured by moving cameras, which appear in many live VR videos, the dynamic background would introduce huge noise and cause the failure of the motion detection algorithm in identifying the user's viewport of interest. As a result, the bandwidth usage for such scenarios would be high, as we verify experimentally in Section 6.2.3. The *LiveDeep* approach [14] employs deep learning to abstract the deep features from the video content and associate them with the user preference for viewport prediction. However, to achieve a high prediction accuracy, the backbone neural network adopted in *LiveDeep* has a deep structure involving a huge number of weights. As a result, it incurs long delays for the model to update and adapt to new features and user preferences during video scene changes, which frequently occur in live VR streaming as we demonstrate in Section 6.2.2. In *LiveObj*, we overcome the limitations in the existing approaches by conducting real-time semantics-based detection and tracking at the object level, which is able to achieve high accuracy and low bandwidth consumption to warrant an effective live viewport prediction approach.

4 User Viewing Behavior Analysis

In live VR streaming, no historical user data is available for viewport prediction. In other words, we cannot obtain a specific user's viewport of interest by analyzing the behavior of other users who have watched the same video in the past. Therefore, we predict the user viewport by analyzing the video content, with the hypothesis that users tend to watch the meaningful objects in the video that they are interested in. To validate the hypothesis, we design an experiment to analyze the user viewing behavior using an empirical user head movement dataset [27]. In particular, we choose two test videos "Anitta" and "Cooking" from the dataset and detect the objects in the videos to uncover the relationship between the viewports and the video content.



(a) Screenshot of Video "Anitta" (b) Screenshot of Video "Cooking"

Fig. 2. User behavior analysis for videos "Anitta" and "Cooking" from the dataset [27]. The red boxes label the areas of detected objects. (a) The screenshot of video "Anitta": "A","B","C" indicate the dancers A, B and C, and "D" indicates the other people. (b) The screenshot of video "Cooking": "A","B","C" indicate the players A, B and C.

To analyze the user viewing behavior, we first deploy the YOLOv3 object detection algorithm [24] to detect the objects in each video frame. Then, we implement the tracking algorithm from Collins et al. [10] combined with location-based verification to match the objects between frames. After that, we parse the user head movement data and draw the conclusion on which objects the user has been watching during the live streaming session.

In our analysis we mainly focus on two metrics regarding the relationship between the target object and the video watched by the user, spanning both the spatial and temporal domains. In the spatial domain, we analyze the average spatial portion of the target object in the entire video, which is calculated as the size of the target object divided by the size of the frame, averaging over all the frames. In the temporal domain, we analyze the average temporal portion of the target object watched by the user, which is calculated as the time duration that the user's viewport fully covers the target object divided by the total time duration of the entire video. For both the spatial and temporal metrics, we define a special object named "surroundings", which indicates the portion of the video that is not identified as any object by the object detection algorithm.

Table 1. Analysis of the spatial and temporal metrics in Videos "Anitta" and "Cooking" from the dataset [27].

(a)	Video "Anitta"	
Object	Spatial Portion (%)	Temporal Portion (%)
Dancer A	3.3	9.7
Dancer B	9.8	51.8
Dancer C	6.5	11.7
Other people	8.1	11.6
Surroundings	72.3	15.2
(b)	Video "Cooking"	
Object	Spatial Portion (%)	Temporal Portion (%)
Player A	5.2	19.2
Player B	5.2	19.6
Player C	5.2	42.6
Surroundings	84.4	18.6

Figure 2(a) shows the annotated screenshot for video "Anitta" (with the detected objects), and Table 1(a) lists our analysis results with the spatial and temporal metrics for each object. We observe that the three dancers A, B, and C take very small spatial portions of the video (3.3%, 9.8%, and 6.5%, respectively) but significantly larger temporal portions of the user's viewport (9.7%, 51.8%, and 11.7%, respectively). On the other hand, the "surroundings" that takes a large spatial portion of the video (72.3%) takes only a small temporal portion of the user's viewport (15.2%). To confirm our findings, we conduct the same analysis on the video "Cooking", as shown in Figure 2(b) and Table 1(b). The results are consistent with our observations in the "Anitta" video, in that the spatially small objects (e.g., Player C, 5.2%) would take large temporal portion of the user viewport (e.g., Player C, 42.1%), while the spatially large objects (e.g., Surroundings, 84.4%) would occupy small temporal portions of the user viewport (e.g., Surroundings, 18.6%).

The analysis of the above two videos reveal an important observation that the user's viewport of interest (indicated by the temporal metric) is not correlated with the size of the object (indicated by the spatial metric). Instead, the user's viewport is heavily dependent upon the semantics of the objects (i.e., the degree of importance or attractiveness) in the video, which validates our hypothesis that the users tend to watch the meaningful objects that they are interested in. The validated hypothesis provides us with the intuition of conducting viewport prediction based on detecting meaningful objects in the video, which further involves the following two observations: (1) *Observation #1*: We could achieve high prediction accuracy (i.e., more than 80% in both "Anitta" and "Cooking") if we detect all the objects in the video frames and generate the user viewports accordingly; and (2) *Observation #2*: We should also explore a method to predict the user's potential interest in the "surroundings" to further improve the viewport prediction accuracy, as it still takes around 15% to 20% temporal portion of the video. The two observations provide us with a guideline in developing the live viewport prediction mechanism in *LiveObj*. In reference to *Observation #1*, we develop an object detection-based method for viewport prediction. In reference to *Observation #2*, we further develop a reinforcement learning-based approach to infer the user behavior based on real time user feedback.

5 Live Viewport Prediction

In this section we discuss the design of *LiveObj* by comparing it with three baseline methods (i.e., *Basic*, *Velocity* and *Over-Cover*). We analyze the limitations of the baseline methods and gradually switch to the discussion of our proposed method – *LiveObj*.

5.1 Baseline Methods

5.1.1 The *Velocity* Method

One existing approach that could be implemented to support live VR streaming is the *velocity*-based method [9]. It employs the actual user viewport information obtained from the the previous segments to model the velocity of the user head movement and estimate the viewport in the future segments. Assuming a user viewport location vector: $\overrightarrow{L} = [l_1, l_2, l_3 ....., l_M]$, where l_i represents the user viewport location in the i^{th} frame from the previous video segments, and M represents the number of frames in the video playback buffer, the velocity of the viewport can be calculated as:

$$\overrightarrow{V} = \frac{(\overrightarrow{l_2 - l_1}) + (\overrightarrow{l_3 - l_2}) + ... + (\overrightarrow{l_M - l_{M-1}})}{M-1} = \frac{(\overrightarrow{l_M - l_1})}{M-1} \quad (1)$$

Then, the predicted viewport location in the next frame, l_{M+1}, can be calculated as:

$$l_{M+1} = l_M + \overrightarrow{V} \quad (2)$$

5.1.2 The *Basic* Method

Based on the analysis in Section 4, the *Basic* method detects all the objects in the video and uses their center as the center of the predicted viewport. Given the list of k objects in each frame, $\overrightarrow{O} = [o_1, o_2, o_3 ....., o_k]$, where each $o_i (i = 1...k)$ represents the coordinates of the center of the object $o_i = < o_i^{(x)}, o_i^{(y)} >$. Then, the center coordinates of the predicted viewport (C_x, C_y) can be calculated as:

$$C_x = \frac{1}{k}\sum_{i=1}^{k} o_i^{(x)}; \qquad C_y = \frac{1}{k}\sum_{i=1}^{k} o_i^{(y)} \quad (3)$$

5.1.3 The *Over-Cover* Method

Inspired by the *Motion* method [15], which creates an irregular predicted viewport to cover more potential regions, the *Over-Cover* method predicts viewports to cover all the regions that contain objects. We leverage a multi-object detection algorithm [24] to process the frames and detect the objects. Then, each detected object generates a predicted sub-view that shares the same center with the object, and the aggregation of all the sub-views forms the final predicted viewport.

5.1.4 Summary of the Baseline Methods

The baseline methods adopt intuitive strategies to predict viewport by tracking either the user trajectory or the video content, all of which would lead to sub-optimal outcomes that we aim to improve and compare with in this work. The *Velocity* method leverages the user trajectory to predict the viewport. It can adapt to the user preference changes quickly, but the prediction accuracy may significantly worsen when predicting for a longer period of time (e.g., a few seconds). The *Basic* method, although focusing on the video content, may fail to select the correct object under a multi-object scenario. The *Over-Cover* method covers all the potential objects to maintain a high prediction accuracy but would incur a relatively high bandwidth consumption.

5.2 Proposed Method: *LiveObj*

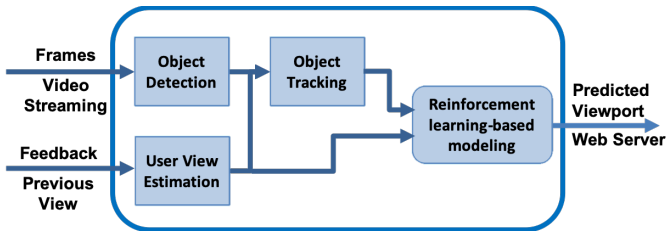


Fig. 3. Workflow of *LiveObj* deployed at the server end.

The *Over-Cover* method is subject to a redundancy problem by incorporating all the detected objects in the predicted viewport while the user could only watch few of them at a time. To address the redundancy problem, we propose to select the most attractive objects based on the user feedback, i.e., the user's current viewport of interest,

to form the predicted viewport. Based on this thought, we develop *LiveObj*, a tracking-based viewport prediction method, which shrinks the predicted viewport from the *Over-Cover* method by filtering out the objects that are less likely to be watched by the user.

Figure 3 illustrates the overall workflow of the *LiveObj* method. *LiveObj* takes two inputs, including the video frames and the user feedback (i.e., the user's head orientations while watching the past video segment). The output of *LiveObj* is the predicted viewport, which includes the selected tiles. The workflow contains four technical steps, namely *object detection*, *user view estimation*, *object tracking*, and *reinforcement learning-based modeling*. The *object detection* step processes the video frames and detects the objects. The *user view estimation* step analyzes the user feedback and estimates the viewport using the *Velocity* method. Then, the *object tracking* step tracks the objects watched by the user. Finally, the tracking results and the estimated user view are fed into the *reinforcement learning-based modeling* step, which updates the status of each tile (i.e., selected or not selected in the predicted viewport). The next subsections describe the technical details of each step.

5.3 Object Detection and Tracking

5.3.1 Object Detection

The accuracy of object detection has been improved dramatically with the invention of convolutional neural network (CNN) [18]. In *LiveObj*, we use the YOLOv3 algorithm [16, 24] to process the video and detect the objects. We employ the pre-trained model [23] capable of recognizing 100 classes of objects, which is sufficient for the prototype system and test benchmarks adopted in this work. For other types of videos that contain objects outside the 100 classes, one can employ additional training dataset and fine-tune the pre-trained model [11].

5.3.2 Object Tracking

The *Over-Cover* method introduces huge redundancy due to covering all the objects. To reduce the bandwidth consumption we need to select the objects that are most likely to be watched by the user from all the detected objects. The intuition behind this objective is that, if the users are currently watching certain objects, it is likely that they will still watch the same objects for a period of time. Therefore, we could select the target objects by tracking the ones that are currently watched.

Different from the object tracking for user analysis in Section 4, the tracking task in the *LiveObj* framework works for live video system and, therefore, the processing must be completed in real time. However, the state-of-the-art multi-object tracking methods, such as feature-mapping [20], mean-shift [33], the Bayesian-based method [32], and the Markov Decision Process (MDP)-based method [28] cannot achieve a real-time performance along with the object detection. Recently, Li et al. [19] proposed a Siamese network-based real-time single object tracking method, which shows the potential of being leveraged for viewport prediction in live streaming system. However, it only supports single object while multi-object scenes are prevalent in VR videos. Besides, in VR videos, we only need to track the objects for a few seconds (i.e., the segment duration), rather than minutes, since the tracking results can be updated by the user feedback periodically. In other words, in every few seconds, we collect the user feedback (i.e., the user head orientations) and further infer which objects the user was previously watching, based on which we update the tracking target. This procedure corrects the tracking failures and adapts to the latest user preference on the video content. Therefore, the tracking errors will not be accumulated but corrected. Also, the periodical user feedback can help with the common scenario in VR streaming where the user switches to a new object due to the change of interest or video scene.

Based on the aforementioned unique VR streaming features and requirements, we develop a fast object tracking method based on the object detection results for short-term multi-object tracking collaborating with the estimated view. Pseudocode 1 shows the tracking algorithm following the object detection in the new frame. Given the object list in the previous user view $OL[K]$ and the detected objects $OLNew[N]$, we calculate the distance between each object in $OL[K]$ and that in $OLNew[N]$ to find the matched objects, which is stored to $D[N]$. Then,

Pseudocode 1 Fast tracking based on object detection.

```
1: procedure IN EACH FRAME(OL[K])                ▷ Object list
2:     OLNew ← N                                 ▷ Detected N objects
3:     for each o[i] ∈ OL do
4:         for each oNew[j] ∈ OLNew do
5:             D[j] ← distance(o[i], oNew[j])
6:         end for
7:         Pairs(i, m) ← m = min(D[N])
8:     end for
9:     for each oNew[j] ∈ OLNew do
10:        if oNew[j] in Esti then
11:            T[j] ← oNew[j]
12:        end if
13:    end for
14: end procedure
```

we find the minimum distance in $D[N]$ as a match for the two objects in different frames, considering that the object would not move by a significantly long distance in a short period of time. The matched objects, represented by their center location, are stored in a pair $< i, m >$ and inserted in the array *Pairs*. Once all the iterations are completed, *Pairs* would contain all the matched objects in the two frames. To further leverage the user trace feedback, we use the estimated user trace to select other candidate objects. Suppose $Esti$ indicates the estimated user view obtained by *user view estimation*, each new object that appears in the predicted viewport, i.e., $OLNew[i] \in Esti$, will be added to the prediction list $T[j]$. Note that in crowded video scenes, a cluster of objects can be treated in the entirety as watched by the user, and our final predicted viewport is generated by the tracked objects and new objects from the estimated view.

5.4 User View Estimation

In the *user view estimation* step, we analyze the user feedback for two purposes: (1) to estimate the user viewport in the future, and (2) to calibrate the current user viewport along with the objects to be tracked. Given the user feedback (i.e., the actual viewport in the past video segment), we first update the latest user viewport and analyze the user head movement pattern, with which we calculate the expected user velocity in the coming frames following Equation (2). Then, we identify the objects that are within the updated user viewport, which are recognized as the objects of interests. These updates are then used by the *object tracking* step for the future segments to improve the prediction accuracy.

5.5 Reinforcement Learning-based Modeling

We note that there are potential errors in the viewport prediction due to object detection failures, as shown in Figure 4.Once the target object is missing from the object detection results in one of the intermediate frames, the object tracking chain would terminate and, consequently, the prediction algorithm would fail to function. Besides, the video scene may change because of the moving camera or the switched views from multiple cameras. As a result, the objects in the previous video scene may disappear in the new scene. Therefore, even a perfect tracking algorithm may fail to predict the user viewport. As a solution, we develop a reinforcement learning-based method to build a user behavior model for each video tile, aiming to minimize the prediction errors.

Our intuition is that different tiles have different probabilities of containing meaningful objects, and the tiles that are more likely to contain meaningful objects are typically more sensitive to object detection errors. We formulate this observation into a policy learning process M presented in Equation (4), where S and A represent the state and action, respectively. $P_{s,a,s'}$ is the probability of choosing action a, given the current state s and the transfer to the new state s'. R represents the reward, also known as the real feedback after the decision has been made, which is used to update $P_{s,a,s'}$ online.

$$M = < S, A, P_{s,a,s'}, R > \tag{4}$$

Table 2. Test videos from public dataset [27].

No.	Video Name	Category	Content	Cameras	Background	Scenery
1	Cooking	Performance	Cooking Show	1	Static	No
2	Front	Sport	Skiing	1	86%	Included
3	Falluja	Documentary	The Fight for Falluja	1	76%	Included
4	Football	Sport	Football Match	1	45%	No
5	Rhinos	Documentary	The Last of the Rhinos	2	66%	Included
6	Korean	Performance	Weekly Idol-Dancing	2	11%	No
7	RioVR	Talk Show	Interview	1	Static	No
8	FemaleBasketball	Sport	Female Basketball Match	1	Static	No
9	Fighting	Sport	SHOWTIME Boxing	2	22%	No
10	Anitta	Performance	Dancing	1	Static	No

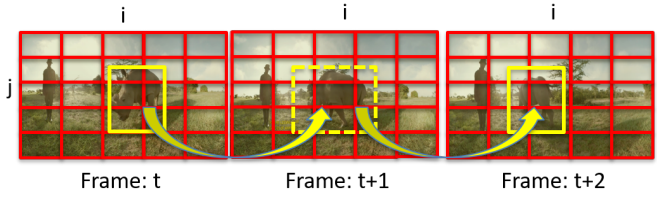


Fig. 4. Error caused by the failure of object detection. Each frame is divided into $i \times j$ tiles indicated by the red edge net. The yellow rectangle indicates the detected object, and the yellow dotted rectangle indicates the failure of detecting the same object in a different frame. The yellow arrow shows the object tracking flow of ground truth. The missing detection in the intermediate frame (i.e., Frame t+1) would break the tracking chain and thus compromise the prediction accuracy.

The goal of the system is to learn the different sensitivity of each tile to the object detection failure by setting different values of $P_{s,a,s'}$. The state-value function is formulated in Equation (5), which indicates the expectation of using different action a given the actual state S_t at time t. The action-value function is formulated in Equation (6), which estimates the value when choosing the action a for all possible states $s \in S$, where γ is a parameter in reward. Then, the final goal is to find the $max(Q_{s,a})$ by calculating each $P_{s,a,s'}$.

$$v = E[Q_{s,a}|S_t = s] \tag{5}$$

$$Q_{s,a} = R_s^a + \gamma \sum_{s' \in S} P_{s,a,s'} v \tag{6}$$

Solving $max(Q_{s,a})$ and calculating $P_{s,a,s'}$ require many iterations and are thus highly time consuming. To address this problem, we employ a modified Q-learning process [26] to solve the optimization problem in a greedy manner. The Q-learning process in this live viewport prediction scenario is unique and different from its traditional applications. First, the predictions are based on both the current input information (i.e., object tracking and view estimation results) and the previous state (i.e., selected or not). Second, the reward is collected online based on the user feedback, and it is changing throughout the video session instead of a pre-set reward matrix R. Third, instead of going through the whole video and calculating the Q values by finding the maximum of the expectation, we must update the Q table for each prediction, due to the unavailability of the video content under the live streaming scenario.

In particular, we create a Q table for each tile. For each Q table, there are four cases, namely "objects only", "objects and viewport", "viewport only", and "no objects or viewport". Combining them with the two previous states "selected" and "not selected", we obtain 8 combinations of options for the state s in each table. For each state, there are two actions (i.e., a), namely "select" and "not select". Therefore, in each table we have 8 states and 2 actions. The reward for each table is updated based on whether the user watched the tile or not. The selection of action a based on s can be turned into the problem of finding $max(Q(s,s'))$ under the same input, where s is the current state and s' is the next state.

Pseudocode 2 Update Q table.

```
procedure IN EACH FRAME(Q[N])
    i ← N
    while i ≠ 0 do
        Update s in Q[i] based on the previous state and input frame
        Choose a by max(Q[i](s,s'))
        Update R based on the user feedback
        Q[i](s,s') = Q[i](s,s') + αR[i]
        i ← i − 1
    end while
end procedure
```

We initialize the Q table to make every tile as selected by assigning a maximum score to the "select" action. Consequently, in the first few frames all the tiles are labeled as "predicted". Then, the Q table is updated following Pseudocode 2 and the chosen action a. In particular, we assign the reward with different values for success and failure (i.e., 3 for success and -2 for failure). Also, we set the maximum and minimum values (i.e., 5 and -3) for the Q table and reset the score to the maximum and minimum if it is out of the range. The learning rate α is set to 0.5.

6 Evaluation Results

6.1 Experimental setup

In the experiments, we use a Dell workstation with two Intel Xeon E5-2623 CPU, one GPU of Titan X and 32GB RAM. We employ the pre-trained YOLOv3 model [23] for object detection and OpenCV for video processing. To evaluate *LiveObj*, we employ 10 VR videos watched by 48 users from a public user head movement dataset [27]. The 10 videos cover multiple types of content, as described in Table 2, including sports, performance, talk show, and documentary videos. Some of the videos are captured by multiple cameras (i.e., Videos 5, 6, and 9) where the video content is switched between the cameras during playback, while the rest of the videos are shot by a single camera. The background in Videos 1, 7, 8 and 10 is static, and the dynamic background in the rest of the videos is caused by either the movement of the camera or the switching between multiple cameras. The estimated portion of video duration with dynamic background over the entire video session is shown in the "Background" column.

6.2 Prediction Accuracy and Bandwidth

6.2.1 Accuracy evaluation metric

We employ a binary metric "match or not match" to evaluate the prediction accuracy for each individual frame, i.e., "match" if the predicted viewport covers the user's actual view and "not match" if otherwise. Then, the overall prediction accuracy of the whole video is defined as the percentage of the frames for which the prediction algorithm achieve a "match" result. In our experiment, the prediction algorithm generates the predicted user view for each frame. Next, we compare it with the

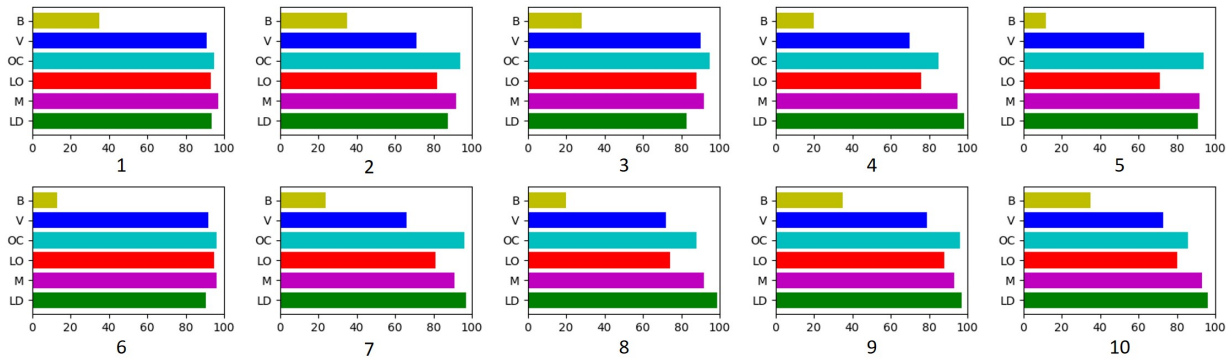


Fig. 5. Prediction accuracy for 48 users watching 10 videos. The X-axis is the prediction accuracy (%), and the Y-axis represents the viewport prediction methods for comparison: "B" – *Basic*, "V" – *Velocity*, "OC" – *Over-Cover*, "LO" – *LiveObj*, "M" – *Motion*, "LD" – *LiveDeep*.

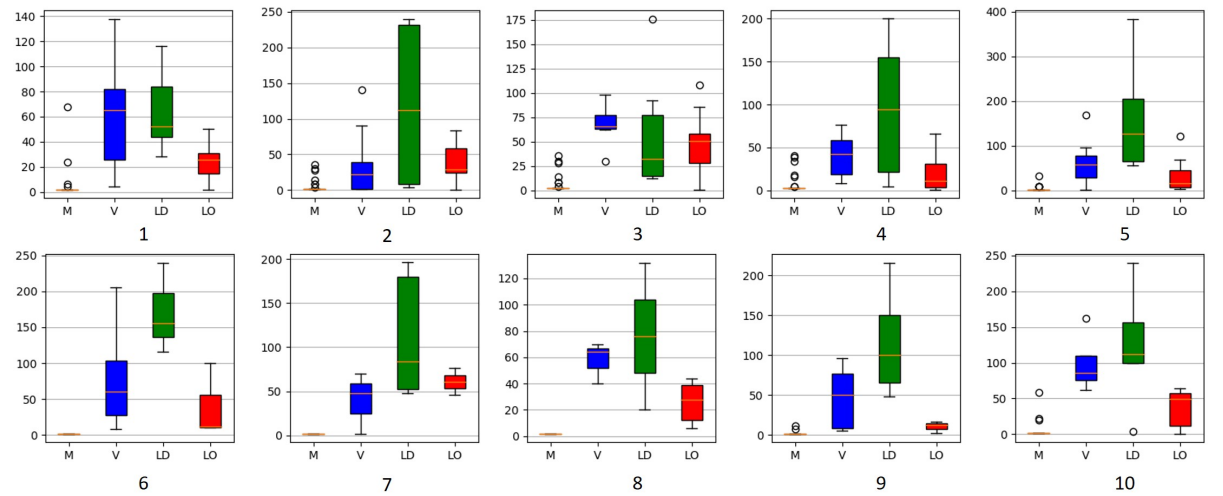


Fig. 6. Comparison of error duration with the 4 methods, *Motion* (*M*), *Velocity* (*V*), *LiveDeep* (*LD*), and *LiveObj* (*LO*) over the 10 test videos, where the Y-axis indicates the number of consecutive frames that the error lasts for.

ground truth (i.e., the actual user view for this frame) obtained from the dataset to draw a conclusion of either "match" or "not match". After processing all the frames in the video, we count the number of frames that have a "match" result and calculate its percentage over the total number of frames as the prediction accuracy.

6.2.2 Accuracy Evaluation Results

We evaluate the prediction accuracy of *LiveObj* in comparison with the three baseline methods discussed in Section 5.1, namely *Velocity*, *Basic*, and *Over-Cover*, plus the two state-of-the-art live viewport prediction approaches *Motion* [15] and *LiveDeep* [14].

Average Prediction Accuracy. Figure 5 shows the average prediction accuracy results for all the frames in the test videos. In a nutshell, *Over-Cover* (*OC*), *LiveDeep* (*LD*) and *Motion* (*M*) achieve the highest prediction accuracy, which is around 90%. Our *LiveObj* achieves similar accuracy in Videos 1, 3, 6 and 9. For the other videos, the accuracy of *LiveObj* is all higher than 70% with most cases higher than 80%. Although the *Velocity* (V) method could achieve a high prediction accuracy in Videos 1, 3 and 6, the performance is not stable, varying from 60% to 80% in the rest of the videos. The accuracy for the *Basic* (*B*) method is lower than 50% for all the test videos.

We further analyze the content of videos 2 and 5, for which *LiveObj* achieves a lower prediction accuracy, and we observe that the two videos are mostly outdoor that contain natural scenes. In such cases, the object detection may fail to capture the mountains, houses, and several other objects due to the limitation of the prediction model. In addition, users tend to watch the surroundings more often as the scene is constantly changing. Although Video 3 is also outdoor, it contains the urban battlefield that the users are more likely to focus on for a longer duration. As a result, the prediction accuracy for Video 3 is higher than Videos 2 and 5. Another factor that affects the prediction accuracy is the distortion in the original video frames, which could make the object detection algorithm fail to generate accurate results. For example, in Video 5, the 360-degree camera is mounted on the back of the target animal, which makes the view of the animal itself distorted in the original video frames and thus hard to be detected. Videos 6 and 10 both contain multiple interesting objects. However, the objects in Video 10 are located in different orientations, while those in Video 6 are rather concentrated. As a result, the users are more likely to switch among these objects in Video 10 during the video session, which reduces the prediction accuracy. Video 4 also achieves a lower accuracy because the scene changes between indoor and outdoor frequently, and the users would look around when it changes. Despite the various complexities and challenges, *LiveObj* still stays above 80% of accuracy for most of the videos given the real-time user feedback mechanism we adopted to compensate for the potential prediction errors. Also, the major advantage of *LiveObj*, as compared to *Over-Cover* and *Motion*, lies in its premium bandwidth usage (discussed in Section 6.2.3) while still maintaining acceptable prediction accuracy. In addition, the minor difference in prediction accuracy can be compensated by the error recovery strategies as discussed in Section 6.3.

Error Duration. The average accuracy results presented in Figure 5 only show the overall macro-prediction performance. Since the individual prediction errors may compromise the instant user experience, we further analyze the micro-prediction performance by checking how fast each method could recover from the individual errors. More specifically, we count the number of consecutive frames with wrong predictions as the *error duration*, which indicates the time duration that the prediction error lasts for. Figure 6 shows the error duration results of *LiveObj (LO)* in comparison with *Motion (M)*, *Velocity (V)*, and *LiveDeep (LD)* over all the frames in the 10 test videos, where the Y axis indicates the error duration, and the X axis indicates the different viewport prediction methods. The results indicate that *Motion* and *LiveObj* achieve much lower error durations than *LiveDeep* and *Velocity*. It is because (1) *LiveDeep* involves large-size and computation-intensive deep learning models, while *Motion* and *LiveObj* only require lightweight computer vision or user preference models; and (2) *Velocity* is challenged by the frequent and arbitrary user head movement patterns that are hard to be accurately modeled by the velocity. In *LiveObj*, the object detection and training are based on pre-trained, lightweight models and thus adaptive to the scene switches. Therefore, *LiveObj* demonstrates superior error duration compared to *LiveDeep* and *Velocity*. On the other hand, the error duration of *LiveObj* is higher than *Motion*, indicating that *Motion* is even more adaptive to prediction errors; however, the simplicity of *Motion* would result in lower bandwidth savings, as discussed in Section 6.2.3.

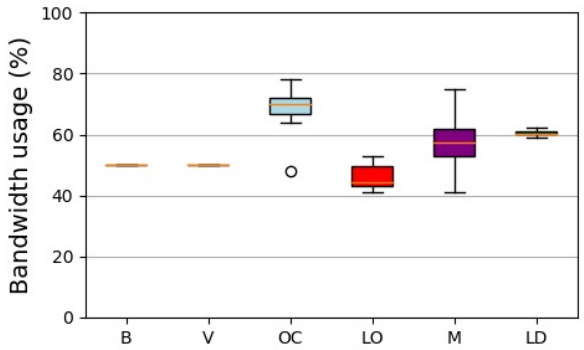


Fig. 7. Average bandwidth usage over the 10 test videos. The X-axis represents the viewport prediction methods for comparison: "B" – *Basic*, "V" – *Velocity*, "OC" – *Over-Cover*, "LO" – *LiveObj*, "M" – *Motion*, "LD" – *LiveDeep*.

6.2.3 Bandwidth Evaluation Results

We evaluate the average bandwidth usage for all the methods, over the 48 users and 10 videos with 2-second buffer length, in comparison to the original bandwidth usage. The results are shown in Figure 7. We observe that the *Over-Cover (OC)* and *Motion (M)* methods introduce high bandwidth consumption with wide distributions (40% to 80%). On the other hand, the bandwidth usage of *LiveObj* is below 50% in most cases. The *Basic (B)* and *Velocity (V)* methods consume 50% of bandwidth, which is around the upper bound of *LiveObj (LO)*. *LiveDeep (LD)* consumes around 60% of bandwidth. The results indicate that *LiveObj* could achieve a relatively high accuracy with a relatively low bandwidth consumption, well balancing the requirements for accuracy and bandwidth in live VR streaming.

Furthermore, we conduct a micro-bandwidth evaluation of the prediction size for each frame to further analyze the bandwidth consumption of *LiveObj* over time, in comparison with *Motion*. Figure 8 shows the results of 4 representative videos (i.e., Video 1, 3, 6 and 8 watched by User 2). The Videos 1 and 3 are the most representative cases, where *Motion* consumes higher or similar bandwidth compared to *LiveObj*. In Video 1, the prediction size of *Motion* in each frame is around 60%, and it is around 40% for *LiveObj*. In Video 3, both methods have round

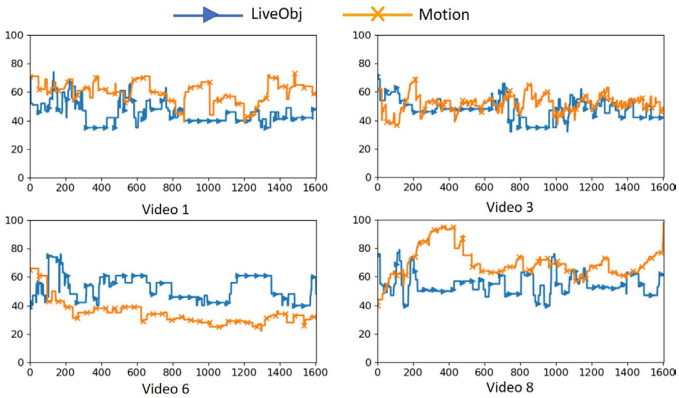


Fig. 8. Micro-bandwidth usage of *LiveObj* and *Motion* over 4 test videos. The X-axis shows the frame number in the video, and the Y-axis shows the bandwidth usage (%).

50% prediction size of the original frame. Video 6 shows one extreme case where *LiveObj* incurs larger prediction size than that of *Motion*. However, in this case, the prediction size by *LiveObj* is still lower than 60% for most of the time. Video 8 shows another extreme case where *Motion* results in large prediction size even close to 100% for some periods, which is not acceptable considering the goal of saving bandwidth. On the contrary, *LiveObj* maintains a low bandwidth usage and is consist with the results shown in Figure 7 with a narrow distribution.

6.2.4 Summary of Accuracy and Bandwidth Evaluations

Descriptive Statistical Analysis. We observe that the *Over-Cover* and *Motion* methods have large variations in bandwidth, and the peak bandwidth consumption is higher than 70%, which is not acceptable given that the original goal of viewport prediction is to reduce bandwidth consumption. For prediction accuracy, the *Basic* and *Velocity* methods obtain the lowest accuracy of 24% and 64%, respectively, while the accuracy of the other methods is all higher than 70%. When looking into the error duration, *LiveDeep* often takes 3 seconds to recover from the prediction errors due to the large model weight. In summary, *LiveObj* achieves a low and narrow distribution of bandwidth consumption compared to *Over-Cover* and *Motion*. Also, the prediction accuracy of *LiveObj* is high and stable compared to the *Basic* and *Velocity* methods. In addition, benefiting from the lightweight user preference model, the error duration of *LiveObj* is much smaller than *LiveDeep*. Overall, *LiveObj* achieves premium performance in all the three evaluation metrics. Even though it may not be the best one under each individual metric, it is the only live viewport prediction approach that achieves well acceptable performance in all the three metrics combined, which is critical for the performance and quality of live VR streaming.

Inferential Statistical Analysis. We further conduct inferential statistical analysis by calculating the confidence interval for the mean values of the prediction accuracy, bandwidth consumption, and error duration results. The detailed results are shown in Table 3. With a confidence level of 0.95, the estimated average prediction accuracy of *LiveObj* ranges from 78% to 87%, which is much better than that of *Basic* and *Velocity*. The estimated average values of bandwidth usage for *LiveObj* are in the range of 43% to 48%, which is the lowest among all the methods. The estimated average error duration of *LiveObj* is between 21 to 40 frames, which indicates that the prediction errors would be corrected in a timely manner. The inferential statistics further confirm our observations in the descriptive statistics that *LiveObj* is the only live viewport prediction approach that achieves well acceptable performance in all the three metrics combined.

6.3 Error Recovery Strategies

There are three recovery strategies when the prediction is incorrect: (1) Selective streaming [25], which delivers a low resolution video for the regions outside the predicted region; (2) Re-transmission, which

Table 3. Inferential statistics: Confidence intervals of the mean values of accuracy, bandwidth usage, and error duration, corresponding to the descriptive statistics in Figures 5, 6, and 7. The significance level is 0.05, and the confidence level is 0.95.

Method	Accuracy (%)	Bandwidth Usage (%)	Error Duration (frames)
Basic	[20, 31]	[50,50]	[79, 125]
Velocity	[70, 81]	[50,50]	[49, 65]
Over-Cover	[90, 95]	[63, 73]	[11, 20]
LiveObj	[78, 87]	[43, 48]	[21, 40]
Motion	[92, 94]	[51, 63]	[7, 11]
LiveDeep	[90, 96]	[60, 61]	[71, 113]

re-transmits the correct viewport from the server to the client; and (3) No recovery, which takes no actions to recover the prediction error. Obviously, Strategy (2) would cause video freezes and eventually increase the bandwidth usage. Note that Strategy (1) is a common practice for the viewport prediction approaches (e.g., *Velocity, Motion, LiveObj, and LiveDeep*) to handle erroneous predictions that would pose significant impact to the user experience. Specifically, this strategy further benefits the performance of *LiveObj* by reducing the actual impact caused by the slight accuracy disadvantage of *LiveObj* as compared to *Motion*.

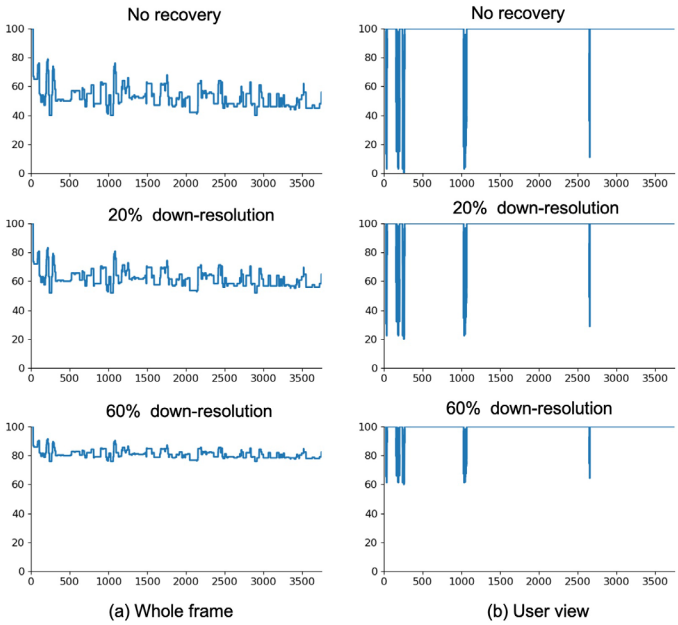


Fig. 9. The bit rate ratio (%) for (a) the whole frame and (b) the actual user view under different error recovery strategies. The X-axis is the frame number, and the Y-axis is the bit rate ratio (%) compared to the original bit rate.

To further demonstrate the difference in performance between Strategies (1) and (3), we analyze the bit rates of the whole frame and the actual user view delivered by *LiveObj*. The results are shown in Figure 9, where the X-axis represents the video frame number, and the Y-axis represents the ratio (%) of the bit rate compared to the original ground-truth version without prediction errors. In particular, Figure 9(a) and (b) show the bit rate results corresponding to the whole 360-degree frame and the actual user's viewport, respectively. In each case, there are 3 sub-figures representing Strategy (3), Strategy (1) with down-resolution rate of 20%, and Strategy (1) with down-resolution rate of 60%. In Figure 9(b), the bit rate ratio of the user view is 100% when the prediction is correct, and the abrupt drops of the bit rate ratio indicate prediction errors. When using Strategy (3), the user could have a blank view where the ratio is almost 0%. When using Strategy (1) with a down-resolution rate of 20%, the user could get a ratio of 20% instead of 0% in the worst case. It is obvious that, increasing the resolution for the non-selected area in a frame could improve the user experience when errors occur. However, the the overall bandwidth savings for the whole frame also decrease as shown in Figure 9(a). The results suggest that, given the high accuracy of *LiveObj*, we can adopt Strategy (3) or Strategy (1) with a low resolution (e.g., 20% of the original). Moreover, in real cases we could adopt different low resolutions for different network capacities to balance the user experience and the bandwidth savings.

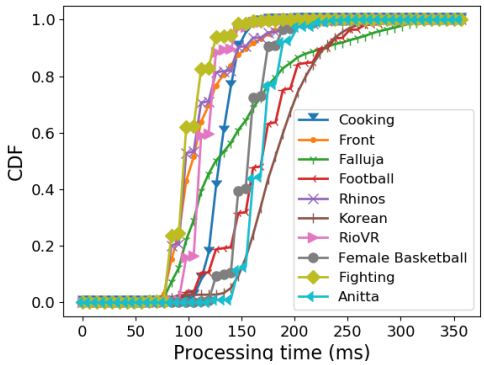


Fig. 10. Cumulative distribution of the average per-frame video processing time over the 10 test videos by using the *LiveObj* method. The X-axis represents the processing time (ms), and the Y-axis represents the cumulative distribution of the processing time values.

6.4 Processing Delay

We further evaluate and analyze the *processing delay* caused by *LiveObj*. Figure 10 shows the cumulative distribution of the average per-frame video processing time over the 10 test videos by using the *LiveObj* method. For each video, we collect the processing time values in the form of 3750 sample points and generate the distribution. We can observe that the processing time for all the frames is less than 350 ms, and it ranges between 75 ms and 200 ms for most of the frames. To achieve real-time performance, we could sample two frames per second from all the original video frames for viewport prediction using *LiveObj*. Then, we leverage the prediction results from these two frames to predict the user viewport for the one second of video. Therefore, the overall processing delay for viewport prediction is around 700 ms per second of video, which means that the viewport prediction can be completed during the video playback of the current segment without delaying the next segment, supporting a smooth streaming experience. Note that the previous accuracy and bandwidth evaluations (i.e., Section 6.2) were carried out on all the frames in the test videos. Now with processing only 2 frames/second for viewport prediction, we re-evaluate the accuracy, error duration, and bandwidth and compare them to the original all-frame case, the average results of which (over 10 test videos) are shown in Table 4. The results reveal that the 2 frames/second case, while meeting the real-time requirement, performs almost equally well in the three evaluation metrics as compared to the all-frame case.

6.5 Impact of Internal Parameter Settings

In this subsection, we conduct further experiments to evaluate the impact of internal algorithms (e.g., the reinforcement learning algorithm) and parameter settings (e.g., learning rate and number of tiles) on the performance of *LiveObj*.

Table 4. Comparison of average viewport prediction performance (over the 10 test videos) in the all-frame and 2 frames/sec cases.

Framerate	Accuracy (%)	Error Duration (frames)	Bandwidth Consumption (%)
All frames	82.84	30.52	45.51
2 frames/sec	81.63	30.48	45.49

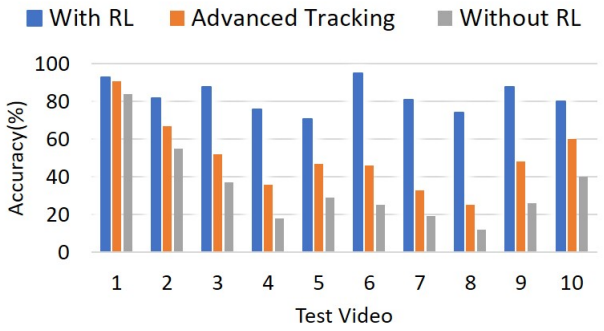


Fig. 11. Comparison of viewport prediction accuracy with/without reinforcement learning and with advanced tracking for the 10 test videos.

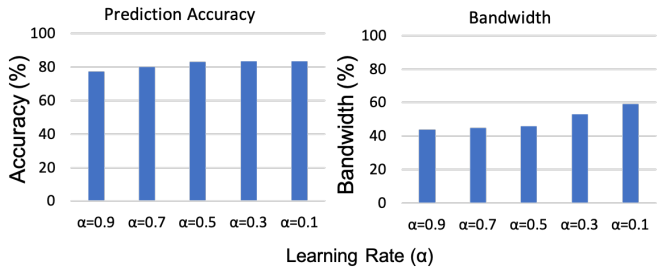


Fig. 12. Average performance of *LiveObj* using different learning rates.

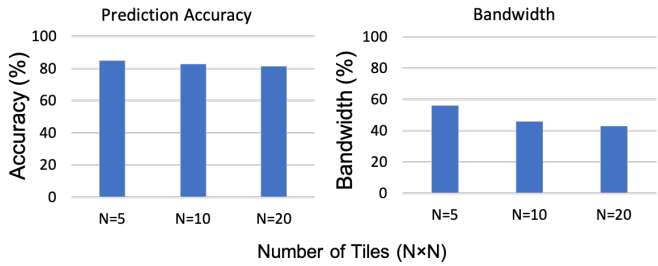


Fig. 13. Average performance of *LiveObj* using different numbers of tiles.

6.5.1 Impact of Reinforcement Learning

Figure 11 shows the viewport prediction accuracy results of the three methods (1) *with reinforcement learning (RL)*; (2) without RL but with Kalman filtering to improve the tracking algorithm (i.e., *Advanced Tracking*); and (3) *without RL*. It is obvious that using RL helps achieve significantly higher accuracy than the other two approaches. Our decision to adopt RL did not come right in the beginning but was following exactly this series of attempts and experiments on related approaches without RL (i.e., *without RL* and *Advanced Tracking*). In particular, our LiveObj first started with a basic object detection and tracking based prediction method (i.e., *without RL*). In this version, we observed the issue of failing to detect the same object in all the frames, which further induces the failure of tracking the object. Therefore, we improved our method with Kalman filtering to reduce the noise (i.e., *Advanced Tracking*). Then, we further noticed the errors caused by failing to cover the viewports that do not contain any objects, which motivated us to develop a tile-based RL method (i.e., *with RL*).

6.5.2 Impact of Learning Rate and Number of Tiles

Figure 12 shows the impact of learning rate (i.e., $\alpha = 0.1, 0.3, 0.5, 0.7$ and 0.9) on prediction accuracy and bandwidth. A higher learning rate could update the tile more often and reduce the bandwidth consumption. However, due to the user's non-smooth movement, more movement noise will be turned into errors, which reduces the prediction accuracy. Overall, the results indicate that the impact of learning rate is not significant on accuracy and bandwidth. In *LiveObj*, we adopt learning rate 0.5, which achieves best results in bandwidth savings and accuracy. Figure 13 shows the impact of the number of tiles (i.e., $N \times N$). It can be observed that changing the number of tiles does not obviously impact the *LiveObj* performance, especially between N=10 and N=20. In *LiveObj*, we adopt N=10.

7 Limitations and Future Work

There are still limitations in the current version of *LiveObj* that can be further improved in the future work. First, it is challenging to predict the user's view switching between objects within one video segment duration, which has not been fully addressed in *LiveObj* and thus it impacts the prediction accuracy. The potential future solution could be leveraging other cues, such as the audio/speech in the videos, to predict the user preference change within one video segment. Secondly, the distortions in the video also impact the prediction accuracy by failing the object detection. This limitation could be addressed by using other projection methods (i.e., cubic projection [13]) or using computer vision-based techniques to tune the camera view. Third, we have not investigated the scalability issue, which may prevent *LiveObj* from serving a large number of users. We plan to explore the scalability issue in the future work by developing an end-to-end VR streaming system incorporating *LiveObj*. Last but not least, the new generation of wireless communication technologies (e.g., 5G) are expected to improve the bandwidth capacity significantly. It is worth exploring how the viewport prediction technique, the advanced wireless communication technology, and the popular VR streaming systems could be integrated to jointly address the bandwidth challenge and meet the quality requirement.

8 Conclusion

This paper targets the bandwidth optimization problem in live VR streaming via viewport prediction. We for the first time analyzed the relationship between the user viewing behavior and the video content in live VR streaming. Our findings reveal the fact that the users spend most of the time watching the meaningful video objects, based on which we proposed a new object semantics-based live viewport prediction framework, namely *LiveObj*, to optimize the bandwidth. *LiveObj* detects and tracks the identifiable and meaningful objects in the VR video and converts the object semantics into well-inferred user viewing behavior based on the user feedback. We evaluated *LiveObj* using 10 VR videos watched by 48 users from a public head movement dataset. The results show that *LiveObj* could obtain a high prediction accuracy while only consuming around half of the original bandwidth. More importantly, the processing delay of *LiveObj* can be kept at an acceptable range suitable for live video streaming. The project repository of *LiveObj* is located at `https://github.com/hwsel/LiveObj`.

Acknowledgment

We appreciate the constructive review comments provided by the anonymous reviewers. This work was partially supported by the National Science Foundation under award CNS-1910085.

REFERENCES

[1] Microsoft Hololens 2. 2019. https://www.microsoft.com/en-us/hololens.

[2] NBA 2019-20 games available in Oculus Venues and NextVR. 2019. https://uploadvr.com/nba-games-oculus-nextvr/.

[3] CNNVR. 2020. https://www.cnn.com/vr.

[4] HTC Vive Cosmos Elite. 2020. https://www.vive.com/us/product/vive-cosmos-elite/overview/.

[5] Oculus Quest 2. 2020. https://www.oculus.com/quest-2/.

[6] Sky Worlds to show Premier League matches live in virtual reality. 2020. https://www.skysports.com/football/news/11095/12158210/sky-worlds-to-show-premier-league-matches-live-in-virtual-reality.

[7] YouTube virtual reality channel. 2020. https://www.youtube.com/channel/UCzuqhhs6NWbgTzMuM09WKDQ.

[8] Y. Ban, L. Xie, Z. Xu, X. Zhang, Z. Guo, and Y. Wang. Cub360: Exploiting cross-users behaviors for viewport prediction in 360 video adaptive streaming. In *IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6, 2018.

[9] Y. Bao, H. Wu, T. Zhang, A. A. Ramli, and X. Liu. Shooting a moving target: Motion-prediction-based transmission for 360-degree videos. In *IEEE International Conference on Big Data (BigData)*, pages 1161–1170, 2016.

[10] R. T. Collins. Mean-shift blob tracking through scale space. In *IEEE conference on Computer Vision and Pattern Recognition (CVPR)*, pages 234–240, 2003.

[11] D. Curro. Borrowing weights from a pretrained network. 2016. https://github.com/BVLC/caffe/wiki/Borrowing-Weights-from-a-Pretrained-Network.

[12] C.-L. Fan, J. Lee, W.-C. Lo, C.-Y. Huang, K.-T. Chen, and C.-H. Hsu. Fixation prediction for 360° video streaming in head-mounted virtual reality. In *ACM Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, pages 67–72, 2017.

[13] Y. Fan, Y. Jin, Z. Meng, and X. Zeng. Pixels and panoramas: An enhanced cubic mapping scheme for video/image-based virtual-reality scenes. *IEEE Consumer Electronics Magazine*, 8(2):44–49, 2019.

[14] X. Feng, Y. Liu, and S. Wei. LiveDeep: Online viewport prediction for live virtual reality streaming using lifelong deep learning. In *IEEE International Conference on Virtual Reality and 3D User Interfaces (VR)*, pages 800–808, 2020.

[15] X. Feng, V. Swaminathan, and S. Wei. Viewport prediction for live 360-degree mobile video streaming using user-content hybrid motion tracking. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies (IMWUT)*, 3(2), 2019.

[16] A. Kathuria. A pytorch implementation of the YOLO v3 object detection algorithm. 2018. https://github.com/ayooshkathuria/pytorch-yolo-v3.

[17] E. Kuzyakov, S. Chen, and R. Peng. Enhancing high-resolution 360 streaming with view prediction. Facebook Inc., 2017. https://engineering.fb.com/2017/04/19/virtual-reality/enhancing-high-resolution-360-streaming-with-view-prediction/.

[18] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 396–404, 1990.

[19] B. Li, J. Yan, W. Wu, Z. Zhu, and X. Hu. High performance visual tracking with siamese region proposal network. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8971–8980, 2018.

[20] C. Liu, J. Yuen, and A. Torralba. SIFT Flow: Dense correspondence across scenes and its applications. *IEEE transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 33(5):978–994, 2011.

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

[22] F. Qian, L. Ji, B. Han, and V. Gopalakrishnan. Optimizing 360 video delivery over cellular networks. In *ACM Workshop on All Things Cellular: Operations, Applications and Challenges*, pages 1–6, 2016.

[23] J. Redmon and A. Farhadi. YOLO: Real-time object detection. 2017. https://pjreddie.com/darknet/yolo/.

[24] J. Redmon and A. Farhadi. YOLOv3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[25] J. Son, D. Jang, and E.-S. Ryu. Implementing 360 video tiled streaming system. In *ACM Multimedia Systems Conference (MMSys)*, pages 521–524, 2018.

[26] C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[27] C. Wu, Z. Tan, Z. Wang, and S. Yang. A dataset for exploring user behaviors in vr spherical video streaming. In *ACM Multimedia Systems Conference (MMSys)*, pages 193–198, 2017.

[28] Y. Xiang, A. Alahi, and S. Savarese. Learning to track: Online multi-object tracking by decision making. In *IEEE International Conference on Computer Vision (ICCV)*, pages 4705–4713, 2015.

[29] M. Xu, Y. Song, J. Wang, M. Qiao, L. Huo, and Z. Wang. Predicting head movement in panoramic video: A deep reinforcement learning approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 41(11):2693–2708, 2018.

[30] Y. Xu, Y. Dong, J. Wu, Z. Sun, Z. Shi, J. Yu, and S. Gao. Gaze prediction in dynamic 360 immersive videos. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5333–5342, 2018.

[31] Z. Xu, X. Zhang, K. Zhang, and Z. Guo. Probabilistic viewport adaptive streaming for 360-degree videos. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, 2018.

[32] J. H. Yoon, M.-H. Yang, J. Lim, and K.-J. Yoon. Bayesian multi-object tracking using motion context from multiple objects. In *IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 33–40, 2015.

[33] H. Zhou, Y. Yuan, and C. Shi. Object tracking using SIFT features and mean shift. *Computer Vision and Image Understanding*, 113(3):345–352, 2009.

[34] Z. Zivkovic. Improved adaptive Gaussian mixture model for background subtraction. In *International Conference on Pattern Recognition (ICPR)*, volume 2, pages 28–31, 2004.