DeepGame: Efficient Video Encoding for Cloud Gaming

Omar Mossad
Simon Fraser University
Burnaby, BC, Canada

Khaled Diab
Simon Fraser University
Burnaby, BC, Canada

Ihab Amer
Advanced Micro Devices, Inc.[1]
Markham, ON, Canada

Mohamed Hefeeda
Simon Fraser University
Burnaby, BC, Canada

ABSTRACT

Cloud gaming enables users to play games on virtually any device. This is achieved by offloading the game rendering and encoding to cloud datacenters. As game resolutions and frame rates increase, cloud gaming platforms face a major challenge to stream high quality games due to the high bandwidth and low latency requirements. In this paper, we propose a new video encoding pipeline, called DeepGame, for cloud gaming platforms to reduce the bandwidth requirements with limited to no impact on the player quality of experience. DeepGame learns the player's contextual interest in the game and the temporal correlation of that interest using a spatio-temporal deep neural network. Then, it encodes various areas in the video frames with different quality levels proportional to their contextual importance. DeepGame does not change the source code of the video encoder or the video game, and it does not require any additional hardware or software at the client side. We implemented DeepGame in an open-source cloud gaming platform and evaluated its performance using multiple popular games. We also conducted a subjective study with real players to demonstrate the potential gains achieved by DeepGame and its practicality. Our results show that DeepGame can reduce the bandwidth requirements by up to 36% compared to the baseline encoder, while maintaining the same level of perceived quality for players and running in real time.

CCS CONCEPTS

• **Applied computing** → *Computer games*; • **Computing methodologies** → **Interest point and salient region detections**; • **Information systems** → **Multimedia streaming**.

KEYWORDS

Cloud Gaming, Content-Based Video Encoding

ACM Reference Format:
Omar Mossad, Khaled Diab, Ihab Amer, and Mohamed Hefeeda. 2021. DeepGame: Efficient Video Encoding for Cloud Gaming. In *Proceedings of the 29th ACM International Conference on Multimedia (MM '21), October 20–24, 2021, Virtual Event, China.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3474085.3475594

MM '21, October 20–24, 2021, Virtual Event, China

ACM ISBN 978-1-4503-8651-7/21/10…$15.00
https://doi.org/10.1145/3474085.3475594

1 INTRODUCTION

Cloud gaming (CG) refers to delivering the *game as a video* by offloading game tasks to cloud datacenters. At a high level, a thin client, running virtually on any device, captures the player's actions and sends them to a deployed game server. The server runs the game logic, renders scenes, encodes frames using a video encoder, and streams them to the client. The client decodes and plays the received frames using any of the widely available video decoders.

Cloud gaming has become a fast-growing industry in the last few years, and it is expected to gain a larger market share in the near future [6]. Major IT companies already provide cloud gaming services such as Sony PlayStation Now [34], AMD-enabled Google Stadia [4, 14], Microsoft Xbox Cloud Gaming [30], Nvidia GeForce Now [32], and Amazon Tempo [1]. Large-scale cloud gaming platforms (e.g., Stadia), deliver game streams in real time to millions of players [28]. And unlike common videos, video games contain detailed large-sized objects, various visual effects (e.g., such as light and smoke effects), and complex animations. Thus, video games require higher bandwidth channels to deliver them, compared to regular videos. For example, while Netflix requires 5–6 Mbps to watch HD videos, Stadia requires at least 28 Mbps to play HD games [15]. Furthermore, recent video games are more bandwidth demanding. For example, Battlefield 4 and Counter-Strike have a resolution of up to 4K and a frame rate up to 144 fps. At this large scale, even a small reduction in the bandwidth needed for each game session can yield millions of dollars of savings for cloud gaming providers [7] as well as enable more players to participate in cloud gaming.

Optimizing video encoding in cloud gaming to reduce bandwidth requirements is, however a complex research challenge. This is because video encoders face a fundamental trade-off between the stream quality, needed bitrate, and response delay. Specifically, video games require high bitrates, which increases the network transmission and processing delays. This may negatively impact the player's experience. In addition, video encoders need to consider both the spatial and temporal qualities within each frame and across successive frames since players are highly sensitive to visual distortions, while not producing game streams with highly variable bitrates as such streams could result in buffer overflows and unpredictable playout delays at the player side.

In this paper, we propose a new video encoding pipeline, called DeepGame, for cloud gaming to deliver high-quality game streams *without* codec or game modifications. The main idea of DeepGame

[1]

is that not all regions within a frame are of equal interest to players. However, identifying the relative importance of different regions is complex, and goes well beyond traditional object detection algorithms from the computer vision literature. This is because the importance of objects depends on the game context. For example, in shooter games, a target object such as an enemy character is more important than background objects such as buildings and trees, even if the target is small and has less complex texture.

DeepGame takes a learning-based approach to understand the player contextual interest within the game, predict the regions of interest (ROIs) across frames, and allocate bits to different regions based on their importance. Unlike prior works (e.g., [17, 20, 23]), DeepGame does not need additional feedback from players nor does it modify the existing encoders. Thus, DeepGame is easier to deploy in practice.

This paper makes the following contributions:

- We propose a new video encoding pipeline, called DeepGame, for game servers to reduce the bandwidth requirements of cloud games while maintaining the perceived quality.
- We design DeepGame to learn how players interact with the game and accurately predict ROIs in real time with low processing overheads on servers.
- We collect a new dataset to study how players interact with multiple popular video games by tracking their eye fixations, which is available at [2].
- We implement DeepGame and integrate it in a cloud gaming platform to demonstrate its real-time performance.
- We perform a subjective study to analyze the performance of DeepGame. Our results show that participants consistently rate the quality of the gaming sessions encoded using DeepGame higher than those encoded using the base encoder.
- We evaluate DeepGame and compare its performance against the state-of-the-art video encoder. Our results show that DeepGame reduces the bandwidth requirements by up to 36% without imposing noticeable quality degradation.

2 CHALLENGES AND RELATED WORK

2.1 Challenges of Video Encoding in CG

Players often develop a wide range of skills while playing game levels, including selective and acute attention to details, improved sensor-motor skills, and strategic planning. Therefore, skilled players can quickly observe visual artifacts, which often lead to player dissatisfaction. The video encoder is responsible of producing high-quality streams in real time, and its performance is critical for the success of cloud gaming platforms. In the following, we describe the challenges of video encoding in cloud gaming.

Response Delay. The response delay is the elapsed time between when a player takes an action and when the decoded frames are played on their device. Prior studies showed that the tolerable response delay ranges from 100 ms (for action games) to 150 ms (for slow-paced games) [10, 11]. This is unlike live video streaming which can tolerate delays up to 500 ms [13, 40] and on-demand streaming that can tolerate several seconds of buffering [46]. The response delay has three components: client playout, network delay, and server processing delay. The network delay alone usually takes up to 80% of the tolerable response delay [10], which leaves the game server with an extremely short time to execute and optimize all of its tasks. *Thus, video encoders need to strike a balance between producing high-quality streams and incurring additional delays.*

Perceived Quality. The video encoder takes a bitrate budget as input, and it hierarchically allocates bits to groups of pictures, frames and macro-blocks. This bit allocation depends mainly on the complexity of the rendered frames. The difficulty here is that the encoder cannot know the complexity of future game frames, since it cannot wait for these frames to be rendered. Also, the video encoder needs to consider the spatial and temporal qualities of the rendered frames, such as eliminating blocking and flickering artifacts. *Thus, video encoders need to carefully allocate bits to different areas of the frames while producing high-quality streams.*

Player Heterogeneity. Players play the same game differently based on their skills and experience. For example, in a first-person shooting game, an experienced player may pre-fire their weapon without looking at the target. The same player may interact with the same object within a game differently based on the game state. For example, the ball in a soccer game may be important while the player is on the offense, while it becomes less important when the player is shooting a penalty. *Thus, video encoders need to account for different game contexts and player behaviors.*

Modularity and Deployment Costs. The common practice in cloud gaming is to deploy *unmodified* games to these platforms to be within the margins of planned deployment cost and time to market while allowing for portability across different providers. In addition, cloud gaming typically uses common video codecs that already have hardware support in many processor architectures [3, 21, 33]. Therefore, changing the internals of video codecs is not practical, and it may add substantial deployment costs. *Thus, off-the-shelf video encoders should not be modified, and they should not expect additional inputs or game-specific APIs from the game process to query its state or objects.*

2.2 Related Work

Video Encoding for CG. The recent AV1 encoder [9] offers high coding efficiency and video quality. However, its high computational requirements and the limited number of devices that support it stifle its adoption in cloud gaming. Some major cloud providers are using their own proprietary codecs. For example, Google Stadia uses its proprietary VP9 encoder [31] alongside the widely used H.264 encoder to support a large variety of client devices. The successor of H.264, H.265, is the current state-of-the-art standard codec. H.265 provides up to 50% improvement in bandwidth efficiency compared to H.264 [38, 39]. H.265 still has not seen wide adoption in cloud gaming mostly due to licensing issues. In this paper, we conduct our experiments using the H.264 and H.265 codecs to show the current and potential bandwidth savings that can be achieved by DeepGame on top of them.

ROI-based Video Encoding. Hegazy et al. [17] showed that ROI encoding can reduce the bandwidth requirements in cloud gaming. However, their approach requires specifying the ROIs manually, which requires collaboration from the game developers to provide such information. This is unlike DeepGame, which does not require any manual inputs or collaboration from the game developers. Illahi

et al. [20] presented an encoding method based on eye-tracking devices at clients to identify the ROIs. Using eye-tracking devices, however, introduces an additional processing delay to the system and can only work with compatible client devices.

Kaplanyan et al. [23] proposed a neural network-based codec for 3D and AR contents. This work also assumes the availability of an eye gaze tracker at client. In addition, cloud gaming providers often prefer using standard encoders like H.264 and H.265 to support a wide range of edge devices with limited computational power. In contrast, DeepGame optimizes standard video encoders, does not change the video decoders at the client side, and it does not require eye gaze trackers or any additional hardware at the clients. Thus, DeepGame can easily be deployed in practice.

3 DeepGame

In this section, we start by discussing the design principles of DeepGame and how it operates at a high level. Then, we present the details of various components of DeepGame. Finally, we describe the overheads and limitations of DeepGame.

3.1 Design Principles and Overview

Principles of Designing DeepGame. Our objective is to design a video encoding pipeline that produces high-quality game streams while satisfying the stringent latency requirements and a given bandwidth budget. To achieve this objective, we take a principled approach that leverages the following game-specific insights on player interactions with video games and how modern video codecs are designed.

- **Player Contextual Interest.** Not all game frames and objects within these frames are equally important. Specifically, as players interact with games in various ways, our first insight is to capture the interest of players in various game objects based on their *context* within the frames. Traditional visual saliency algorithms do not capture this context [18], because they usually assign the same importance values to similar objects in the frame. In DeepGame, we model the latent relationship between the player's interest and game objects based on the context.
- **Temporal Correlation of Interest.** We leverage several properties of the human visual system (HVS) and the nature of player interaction with video games in designing DeepGame. First, human eyes often fixate on a single area for tens to hundreds of milliseconds [12, 41], which is called the *fixation* period. Second, the visual acuity of humans decreases exponentially as we move away from the fixation point. This is modeled as an exponential function of the so-called eccentricity angle [43]. Finally, the relationship between players' interests and game objects tends to persist over several frames, because objects do not suddenly change locations in sub-millisecond scales. Therefore, we build a model that predicts the fixation points and considers how a player interacts with a game over successive frames.
- **Modern Encoders are Controllable.** Although existing codec implementations are complex, they provide a relatively straightforward set of APIs to control the encoding parameters and bitrates without modifying the underlying video coding algorithms. For example, the `libavcodec` library, which supports major video codecs such as H.264 [45] and H.265 [38], allows scaling, transcoding, and modifying the picture format and encoding bitrate. In this paper, we control the quantization parameter (QP) of every macro-block in the frame, which is easily doable through APIs and it does not require changing the source code of the video codec.

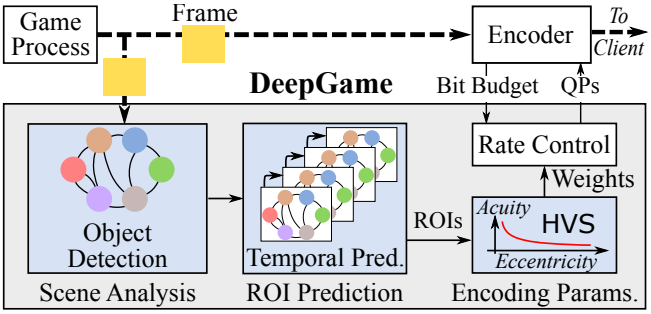


Figure 1: High-level overview of DeepGame.

DeepGame Overview. At high level, DeepGame is a process running between the game process and video encoder process, as shown in Figure 1. The input to DeepGame is the raw game frames generated by the game process, and the output is the encoding parameters (QPs), which are used by the video encoder to produce encoded frames to be transmitted to the client. Notice that both the game and encoder processes are left intact, which enables DeepGame to be deployed in real systems.

During a game session, DeepGame understands the game context and optimizes the encoding parameters in real-time. DeepGame consists of three stages: (i) Scene Analysis, (ii) ROI Prediction, and (iii) Encoding Parameters Calculation. These stages operate in *parallel* while forwarding the outputs from one stage to the next. The pipelining of the three stages is a crucial design decision that enables DeepGame to meet the low latency requirements of CG.

The Scene Analysis and ROI Prediction stages, described in Section 3.2, form a spatio-temporal model to identify the locations of ROIs and how they change over time. The last stage, described in Section 3.3, uses the predicted ROIs and characteristics of the human visual system to compute weights for different areas in each frame. These relative weights are intentionally made abstract and independent of the internal design of the video encoder and its rate control mechanism. Thus, DeepGame is general and can support different video encoders. To be concrete, we describe how the weights produced by DeepGame can be used with the current state-of-the-art encoder, H.265 [39], and its λ rate controller [24, 44].

3.2 Spatio-Temporal Prediction of ROIs

The proposed spatio-temporal model for predicting ROIs is summarized in Figure 2. It starts with a Scene Analysis stage, where the objects in individual game frames are identified. Objects of multiple successive frames are then fed into the second stage, which models the context and relationship among these objects and uses this information to predict areas in the frames that players will likely

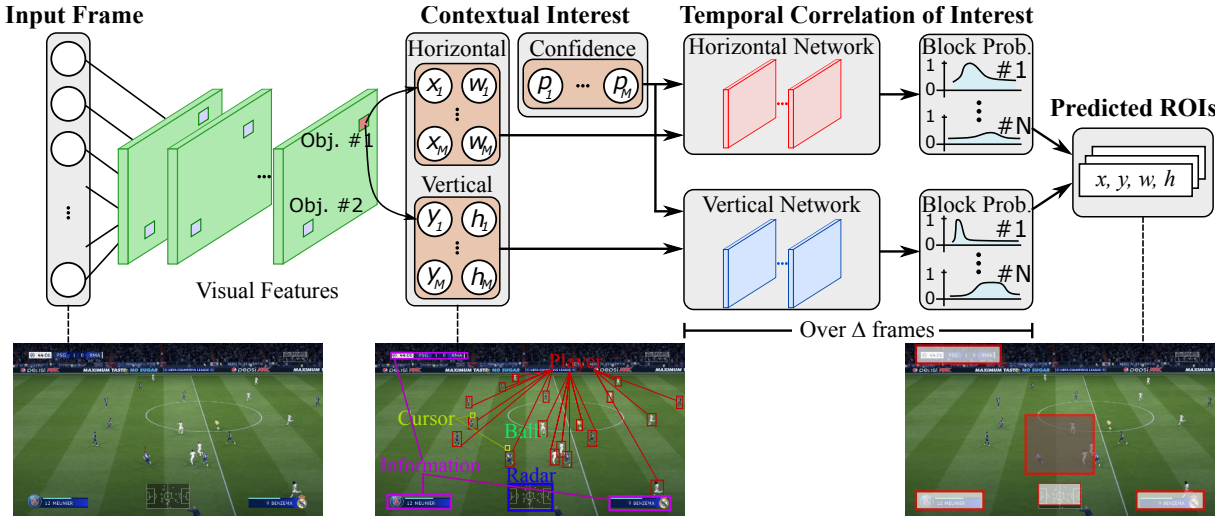


Figure 2: The proposed spatio-temporal ROI prediction neural network.

focus on in the following short period of time. The details of each stage are presented in the following.

Scene Analysis. This stage analyzes the visual features in a given game frame to obtain knowledge about its contents. Although several object detection neural network models have been proposed in the literature (e.g., [25, 27, 37]), most of them are fairly complex and their inference speeds are slow, and thus they cannot satisfy the low latency requirement in CG.

We propose a light-weight object detection model based on Yolo [36]. Our model produces sufficient accuracy for CG and runs in real time. The model is composed of 13 convolutional layers with maxpooling and skip connections. The backbone of the network is Darknet [35] that we train using the COCO dataset [26] and fine tune to identify the most important game objects. The input to our Scene Analysis model is a single frame. The model outputs each identified object in the frame, along with its coordinates, width and height, as well as the class/type of that object, as shown in the second image from the bottom left of Figure 2.

In video gaming, players occasionally glimpse at certain areas in the frames, such as the game map/radar and current score, which we refer to as *auxiliary areas*. This is in addition to the main ROIs which players focus on most of the time. Despite being looked at for short periods of time and thus would unlikely be included in the main ROIs, auxiliary areas are important for the actual playing of the game. Therefore, the visual quality of auxiliary areas should be preserved. In our Scene Analysis model, we define and output special classes for the auxiliary areas, which are later recognized by the last stage where we assign weights to different areas.

ROI Prediction. To consider the temporal changes in the game context, we design a recurrent neural network model. Specifically, we design a Long-Short-Term-Memory (LSTM) network that captures the dynamics of the game context over Δ frames, where Δ is in the order of 5–10 frames sampled within a short period of time. The hidden state in the LSTM network models the effect of the player's actions on the relative importance of different areas across successive frames. For example, moving the cursor from a

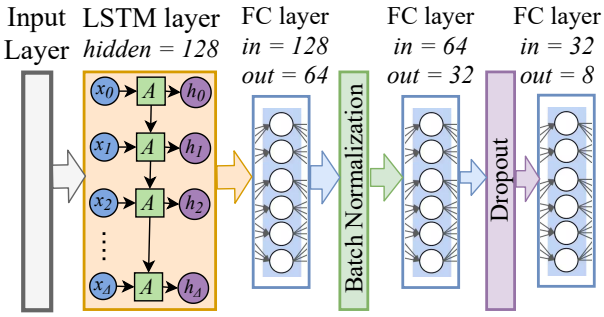


Figure 3: The structure of the LSTM network.

location to another may mean that the ROI has shifted. In addition, the output evolves over time based on the past memory, and the hidden state will be formed based on a short-range memory of the player's past behaviour.

To identify and later encode ROIs with higher quality, we divide each frame into $N \times M$ blocks, where each block can be further divided into the smallest encoding unit for the video encoder, which is usually set to 8×8 pixels. Objects in video games can move with different patterns and speeds along the horizontal and vertical directions across successive frames. Thus, as shown in Figure 2, we divide the LSTM network into two branches; one for the horizontal direction and another for the vertical direction. The horizontal (vertical) branch considers the object presence in blocks along the horizontal (vertical) direction, and it predicts the ROIs within the N (M) blocks. The positive entries in both orientations are then combined to produce the final output in the form of an $N \times M$ array of blocks. This division allows a flexible and fast training since we restrict the output space for each branch to N or M binary elements. We note that ROI blocks are contiguous and adjacent within successive frames as the user visual attention cannot move between disjoint areas in a small amount of time.

The structure of the proposed LSTM network is depicted in Figure 3. It has 128 hidden units with an input size depending on N or M and the number of objects in the game. We use two fully connected layers with ReLU activation functions with sizes of 64 and 32, respectively, batch normalization, and a dropout layer. The outputs at every time step are used to predict the importance of each block. The output layer has N or M outputs, one for each block, and it uses a sigmoid function to create a probability value indicating whether a block is an ROI.

To accurately predict the ROI blocks, the network establishes a correspondence between frame contents and the ROI as determined by the player fixation. Therefore, we need to define a suitable loss function that rewards the network when it covers the correct ROI blocks and adds a penalty when the network unnecessarily includes additional blocks and increases the prediction area. We design a variant of the binary cross entropy [29] to be the loss function of the network. The binary cross entropy is widely used for classification tasks because it quantifies the amount of information that the network learns at each step. Traditional binary cross entropy, however, would penalize overly predicted blocks the same way it penalizes the missed ROIs.

In cloud gaming, especially with skilled players, it is better to be more conservative and cover all locations where the player may focus on, even if the network includes some additional blocks in the ROIs. Designating many blocks as ROIs would, however, substantially impact the encoding performance. To address this trade-off, the loss function uses an additional parameter, P, to reduce false negatives, i.e., avoid missing any ROI blocks. The loss function for each orientation is given by:

$$L_{tot} = \{l_1, l_2, .., l_N\}, \tag{1}$$

$$l_i = -[P \times y_i \times \log \sigma(x_i) + (1 - y_i) \times \log(1 - \sigma(x_i))], \tag{2}$$

where x_i and y_i are the predicted probability and exact label for block i, respectively. We set the P to the number of blocks needed to cover the ROIs for all users, which can be estimated by analyzing a few historical game sessions before training the model.

3.3 Calculating Encoding Parameters

The weight calculation module takes as inputs the ROI predictions, and calculates the weights to be used by the rate controller shown in Figure 1. The weight assignment function is based on the behaviour of the human visual system, and is given by:

$$w[x, y] = \begin{cases} e^{\frac{-1}{F_i Q}} & \text{if } b[x, y] \text{ is ROI or auxiliary,} \\ \frac{1}{R} \sum_{i=1}^{R} e^{\frac{-Q d_i}{F_i D}} & \text{Otherwise,} \end{cases} \tag{3}$$

where $b[x, y]$ is the encoding block with coordinates of x and y, Q is a constant scaling factor controlling the desired discrepancy in bit allocation and quality between ROI and non-ROI areas, R is the total number of ROIs and auxiliary information per frame, D is the diagonal of the frame in pixels, F_i is an importance factor $\in (0, 1]$ that is assigned for ROIs and auxiliary information, and d_i is the distance between the block and the nearest center block of the ROI. We set $F_i = 1$ for ROIs, and $F_i = 0.5$ for auxiliary areas.

The rate controller uses the calculated weights, the bandwidth budget, and the used bits to calculate the QP for each block. DeepGame does not dictate a specific rate control algorithm. Thus, DeepGame relies on prior works (e.g., [17]), to calculate the number of bits allocated per frame. Then, it normalizes the weights in Equation (3) to make sure they sum up to 1, and calculates the number of bits per block proportional to its relative weight. As an example, the rate controller in H.265 would calculate its internal parameter, referred to as λ, using the calculated bits per block. Then, it uses the value of λ to estimate the QPs as $K_1 \times \ln(\lambda) + K_2$, where K_1 and K_2 are rate controller constants [24].

3.4 Overheads and Limitations of DeepGame

DeepGame achieves substantial bandwidth savings and improves the visual quality, as shown in section 4 and Section 5.3. DeepGame, however, requires training the proposed deep learning model and running it on servers in real time to achieve these gains. Training the deep learning model is done *offline* and once per each game *type* (not individual game sessions). For example, we train the model on a soccer video game (e.g., FIFA 20), once and whenever there is a major release for that game which may contain different graphics, resolutions, or visual effects. A separate training is needed for other game types (e.g., CS:GO and NBA). Major games are usually released yearly or on even longer time scales. Training the machine learning model also requires collecting datasets. We note that the spatio-temporal nature of the proposed deep learning model allows it to capture the behavior of players with different skill levels and encode this information in the neural (LSTM) network. Thus, our model supports different players and does not need to be customized for individual players.

While training the machine learning models may take hours, running the inference on these models is much faster and it should be done for each gaming session. In Section 4, we show that the inference models of DeepGame can run in real time on a common, low-end, GPU. In addition, we used the ONNX runtime environment [5] to optimize the inference models of DeepGame and run them on CPUs, and our results indicate the DeepGame can also run in real time on CPUs. More code and model optimizations are possible to reduce the processing requirements and are left for future work.

In addition to running the inference of the trained machine learning models, DeepGame needs to calculate the weights for blocks in the frames and maps them QPs as well as run the rate controller of the encoder. These, however, are simple operations that each take take around 1 ms per frame, as reported in Section 4.

Despite its flexibility to support various games without modifications, DeepGame may produce inaccurate ROI predictions in few cases. When a game has almost no identifiable objects, DeepGame may not be able to identify which part of the screen is the exact ROI. For instance, a game like Minecraft, which has pixelated graphics, proves to be hard to identify objects of interest within its frames.

4 IMPLEMENTATION AND SUBJECTIVE EVALUATION

Implementation. We implemented DeepGame and integrated it in GamingAnywhere [19], which is an open-source cloud gaming system. The training of DeepGame is described in Section 5; we use the inference models of DeepGame to conduct the experiments in this section. GamingAnywhere is composed of a multiple modules running in parallel threads and interact with one another through

data exchange over pipes. One of the modules is an RTSP server that streams the encoded video to the client. To encode frames, they are transferred from the GPU buffer to the encoder through a filter module that converts the frame into YUV format. Finally, the controller module is responsible for transferring the client's keyboard and mouse inputs to the server.

GamingAnywhere supports different video encoders. We first tried using the open-source implementation of the recent H.265 encoder (x265). However, because the open-source version is not yet optimized, the decoding speed was slow at the client side and did not result in smooth playback of the games, even before using DeepGame. Thus, we resorted to using the open-source implementation of H.264 (x264), which is still the most common video encoder in practice.

Subjective Study. We conducted a subjective study to analyze the end-to-end performance of DeepGame, especially how players perceive the visual quality and smoothness of the game frames and whether DeepGame injects any noticeable delays that may impact their gaming experience. This subjective study was approved by our University Research Ethics Board. We used the PC versions of two popular games of different nature: FIFA 20 and CS:GO. Both ran on at 1920x1080 and 30 fps. We setup a GamingAnywhere server to serve these two games to clients over the network, where we can control the encoding rate of the produced videos. To avoid overloading the server, we allowed only one client to connect to it and there is only one gaming session at any time.

We invited five participants to play the two games multiple times. The participants had different experience with each of the two games, ranging from novice to expert. Each participant played eight sessions of each game, four of these sessions were encoded using the standard H.264 base encoder and the other four were also encoded using H.264 but with its encoding parameters optimized using DeepGame in real time. The length of each gaming session was about two minutes. The participants were not informed which video encoding method was used in each gaming session. Also, the order of choosing the encoding method was randomized. In addition, we varied the encoding rate from low bandwidth values of 1 and 2 Mbps to medium and high values of 4 and 10 Mbps.

In total, we collected information from 80 gaming sessions from five participants playing two different games under four bitrate values. After each gaming session, the participant was asked to report the Mean Opinion Score (MOS). MOS is a rating from 1 to 5 following the Absolute Category Rating (ACR) [22], where 1 indicates an extremely bad gaming experience with substantial lags, distortions, and artifacts across the frames. Whereas a score of 5 indicates an excellent quality without any noticeable lags or distortions in the video quality.

Table 1 summarizes the results of our experiments. The results show that DeepGame significantly improves the quality perceived by the participants in all scenarios. For example, for the CS:GO game when the bandwidth is 4 Mbps, DeepGame increases the MOS from 3.6 (average–good quality) to 4.8 (very good–excellent quality). That is, in this case, DeepGame achieved an improvement of about 33% in the MOS over the base encoder under the same bitrate. The improvement in the MOS achieved by DeepGame is especially useful in relatively low bandwidth scenarios. For example, DeepGame improved the MOS from 2.4 (poor quality) to 3.4 (average–good quality) for CS:Go with 4 Mbps.

Game	**FIFA20**		**CS:GO**	
Bitrate/Setup	Base	DeepGame	Base	DeepGame
1 Mbps	1.6	2.4	1.4	2.6
2 Mbps	2.6	3.2	2.4	3.4
4 Mbps	3.8	4.2	3.6	4.8
10 Mbps	4.0	4.4	3.8	4.4

Table 1: Results of the subjective study: DeepGame improves the Mean Opinion Score by up to 33% over the base encoder.

DeepGame is expected to achieve even higher gains for recent games that have resolutions of 2K/4K, run at 60/120 fps, and have very rich graphics and illumination effects. These games require very high bandwidth that most current players do not have. Because of its substantial quality improvements when the available bandwidth is smaller than the needed bandwidth for the base video encoder, DeepGame could enable cloud gaming providers to offer recent video games to a larger number of players.

Processing Time of DeepGame. We measure the processing times of different components of DeepGame. Notice that in all of the above experiments, DeepGame was running in real time during the gaming sessions, and the MOS results reported in Table 1 did not indicate any lags perceived by the players. Also, recall that the DeepGame's components are pipelined and run in parallel with the encoding operations, and thus they do not impose additional delays to the encoding pipeline. During the experiments, all games were running at 30 fps, i.e., the allowed time for each frame is 33.33 ms. The spatio-temporal network of DeepGame uses Δ frames as input, however these frames are sampled over a short period of time. Specifically, we sample $\Delta = 10$ frames during the last second of gameplay. According to [16], the gameplay context can be accurately predicted using the data from the last second. Using this sampled approach, DeepGame does not need to run the object detection network on each frame and in our experiments it runs every 3 frames. As result, the ROI prediction time will not add any processing delays to the pipeline as long as it is lower than 100 ms per frame and DeepGame runs every third frame, where the frame rate is 30 fps. We measured the total time for DeepGame to process each frame. The average processing time was 82 ms, with a standard deviation of 1.5 ms. For video games with higher frame rates, a larger sampling period can be employed to ensure DeepGame meet the real time constraint.

The predicted ROIs obtained from the spatio-temporal network are then used to encode the future frames that will occur during a short period of time equivalent to the minimum duration for eye fixations [42]. Thus, all frames that occur during the next 0.2 sec, i.e., 6 frames will have the same encoding weights and the encoder doesn't incur any delays as the weights are readily available from the previous ROI prediction stage.

We further analyze the processing times of the other components of the entire video encoding pipeline, which are weight calculation, rate control, and actual encoding of the video frames. The last two components are common, whether DeepGame is used or not. The

average processing time per frame for the rate control component was 1.158 ms and for the encoding component was 4 ms. Whereas, the average processing time per frame for the additional weight calculation component of DeepGame was only 0.31 ms.

5 OBJECTIVE EVALUATION AND ANALYSIS

In this section, we evaluate the accuracy of the proposed ROI prediction model, relative to the ground truth ROIs that we have collected from four popular games. We then compare the encoding performance of DeepGame against the base encoder using multiple objective metrics.

5.1 Dataset and Model Training

Gaming ROI Dataset. We conduct experiments to collect a dataset showing where players look at while playing different games. Our setup consists of a Gazepoint GP3 eye gaze tracking device, a PC, and a PlayStation 4 (PS4) console. The PC has a CPU with 8 cores running at 4 GHz, 32 GB of RAM, and a GPU with 2,048 cores and 4 GB of memory. We used a 27-inch HD LCD monitor at 60 Hz.

We first ran the calibration software of the eye gaze tracker. Then, during a gaming session, we recorded the game frames and collected the eye fixations of the players using the eye gaze tracking device. We used two PC games and two PS4 games in our study. For the PC games, we recorded the games at full HD resolution. For PS4 games, we mirrored the games to the PC where the eye gaze tracker is installed. The PS4 processor down-scales the frames to 720p as it becomes busy with mirroring and running the game.

The considered four games represent different categories and camera perspectives as follows:[1] (i) *FIFA 20:* is a soccer game where the user controls and switches between players of a single team, (ii) *CS:GO:* is a first person shooter game where the user controls the movements and aim of their character to eliminate enemies, (iii) *NBA Live 19:* is a basketball game where the user controls the five players in their team, and (iv) *NHL 19* is an ice hockey game where the player controls a team of players as if the player is sitting at the goalkeeper side.

We invited eight participants with different skill levels to play each of the four games multiple times. Each participant played each game for an average duration of seven minutes. In total, we recorded information from 98 gaming sessions, for a total playing time of 3 hours and 44 minutes. The games were played at their normal frame rate of 30 fps, but the eye gaze tracker could only capture at 10 fps, that is, the tracker recorded the gaze location every third frame. This is not a major concern as the human visual system cannot switch its fixation in a period shorter than 0.2 sec [42], which is double the duration between the frames captured by the tracker.

The final dataset has 138,426 game frames from four popular games, along with the gaze location in each frame. The dataset will be made public at [2].

Training of DeepGame's Neural Network Models. The training was performed in two steps. First, we fine-tuned the object detection network to extract the game objects and auxiliary information in our dataset frames. Next, we post-processed these detections and used them to train the LSTM neural network. Each object was mapped to the blocks it occupies in an $N \times M$ array of blocks. The training and testing are done using the leave-one-out approach. For each game, we trained using the data of all players except one and tested on the left-out player. Then, we choose another player to leave out, and we calculate the average results across all cases. The learning rate is set to 0.001 and we train the network for 30 epochs. Since the data was collected at 10 fps, a single data sample consists of $\Delta = 10$ consecutive raw frames in .jpg format and the Ground Truth is the ROI blocks in the next 2 frames similar to what we mentioned in Section 4. We mapped these fixation points to the $N \times M$ ROI blocks where $N = M = 8$. The mapping process is straightforward: we draw a circular region around the fixation point and designate the blocks that cover this region as ROIs. The radius of the circle is set to 70 pixels as this value represents the 2° of visual angle that covers the foveal region on the screen [23].

In the following, we compare DeepGame versus a baseline encoder. This is because the state-of-art ROI encoding system, CAVE [17], requires *manual* specifications of ROIs.

5.2 ROI Prediction Accuracy

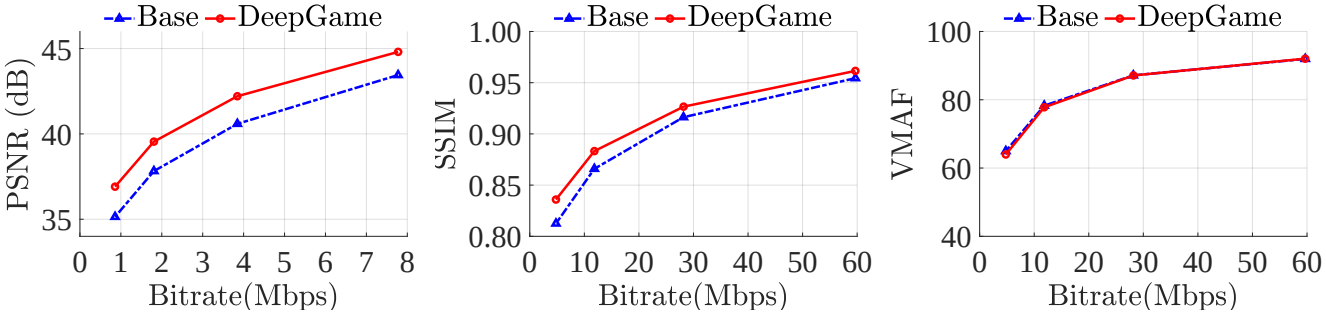
In this section, we assess the performance of the proposed ROI prediction model. We report two important metrics: the prediction accuracy and prediction area. We define the prediction accuracy as the percentage of frames where the model accurately predicted the ROI blocks. This metric determines how well we are able to cover the ROI blocks. The second metric, the prediction area, indicates the additional predicted blocks. The smaller the prediction area, the better the encoding efficiency. The additional predicted blocks, however, are necessary to cover all possible fixations.

Table 2 shows the prediction accuracy and area for each game. The proposed model yields a high accuracy of 85.95% for FIFA 20 while using an ROI area of around 12% of the entire frame. This game has the majority of scenes with distinct objects located across the frame. In addition, the soccer field does not have a lot of details. This increases the model ability to accurately learn the game context and predict the exact ROIs. In CS:GO, a first-person shooter game, the prediction accuracy and area are 82.85% and 9.65%, respectively. Although this is a complex game with faster motion and interactions compared to the other games, the model learns the game context as many eye fixations are located near the center. The model accuracy for NBA19 and NHL19 is 78.17% and 73.96%, respectively. This is because of the wide range of eye fixations in these games. In addition, most of the scenes have a large number of insignificant objects distributed across the frame such as spectators and floor paintings. Therefore, the model predicts more blocks to cover the majority of the ROIs.

5.3 Comparison of DeepGame vs. Base Encoder

We analyze the encoding performance of DeepGame and compare it against the state-of-the-art H.265 video encoder, which is referred to as Base in the figures.

[1] Three of the four games, FIFA 20, NHL 19, and NBA Live 19, and all screenshots taken from them are licensed property of Electronic Arts, Inc. The fourth game, CS:GO, and all screenshots taken from it are licensed property of Valve Corporation. We obtained permissions to conduct experiments with these games and include some of their screenshots in this paper.



(a) PSNR (b) SSIM (c) VMAF

Figure 4: The encoding performance of DeepGame versus the base H.265 video encoder for CS:GO.

Metric/Game	FIFA 20	CS:GO	NBA Live 19	NHL19
Prediction Acc.	85.95%	82.85%	78.17%	73.96%
Prediction Area	11.91%	9.65%	21.80%	20.99%

Table 2: Accuracy of the proposed ROI prediction model.

Segment	BD-Rate: SSIM	BD-Rate: PSNR
FIFA 20 (2K@60)	-33.01%	-35.06%
CS:GO (2K@60)	-23.34%	-19.11%
NBA Live 19 (720@30)	-31.94%	-36.40%
NHL 19 (720p@30)	-20.80%	-22.42%

Table 3: Potential bitrate savings by DeepGame compared to the base encoder computed using the B–D rate metric.

We report the following four metrics as they represent the perceived quality. (1) *The Bjontegaard (B–D rate)* [8]: It computes the average bitrate savings between two encoding methods by calculating the average area between their Rate–Distortion (R–D) curves. A negative value indicates a bitrate saving achieved by the first method compared to the second one. (2) *Peak Signal-to-Noise Ratio (PSNR)*: It represents the visual fidelity of the frames. (3) *Structural similarity index measure (SSIM)*: It measures the similarity between two images. (4) *Video Multimethod Assessment Fusion (VMAF)* [7]: It estimates the subjective video quality based on a reference and distorted video sequence.

We measure the PSNR and SSIM metrics in the ROIs, since they are the main areas where players focus on. The VMAF metric complements the PSNR and SSIM metrics as it evaluates the quality across the whole video sequence.

First, we analyze the bandwidth savings of DeepGame over the entire frame by reporting the B–D rate metric using the SSIM and PSNR R–D curves calculated over the ROIs for all four video games. Table 3 shows the bandwidth savings achieved by DeepGame compared to the base H.265 encoder. Recall that DeepGame needs to predict additional ROI blocks to cover the different user behaviours. Despite these additional blocks, DeepGame outperforms the base encoder across all games and qualities. Specifically, the table shows that DeepGame reduces the bandwidth usage by up to 35% and 19.11% for FIFA 20 and CS:GO, respectively. In addition, DeepGame yields bandwidth savings up to 36% and 22% for NBA Live 19 and NHL 19, respectively.

We next present sample results for the PSNR, SSIM, and VMAF metrics for the CS:GO game in Figure 4. The results for the three other games are similar and omitted due to space limitations. The figure shows that DeepGame achieves higher PSNR and SSIM values compared to the base encoder for all bitrates for both games. The figures also show that both DeepGame and base encoder yield almost the same VMAF values, which are measured across the entire sequence, not just ROIs. This indicates that DeepGame does not introduce distortions despite the bandwidth savings.

6 CONCLUSIONS

Delivery of high-quality games to large-scale players represents a major challenge to cloud gaming platforms due to its high bandwidth requirements. We designed and implemented DeepGame, a new pipeline to efficiently encode video streams for cloud gaming. DeepGame learns the context of the game and predicts the regions of interest (ROIs) in the video frames using a spatio-temporal deep neural network model. DeepGame then encodes the ROIs with a relatively higher quality than other regions, while maintaining a smooth quality within each frame and across successive frames. We have collected a new dataset containing detailed information about 98 gaming sessions from several participants of different skill levels playing four popular video games: FIFA 20, NBA Live 19, NHL 19, and CS:GO. The dataset has about 140K frames with their associated ROIs. The dataset and the code of DeepGame are publicly available for other researchers. We implemented DeepGame in an open-source cloud gaming platform and conducted a subject study to show its practicality and performance. The results show that DeepGame can improve the quality of the gaming experience, measured the Mean Opinion Score (MOS), by up to 33%. In addition, we demonstrated the potential bandwidth savings of DeepGame compared to the state-the-art H.265 (HEVC) video encoder using multiple objective metrics: BD-rate, SSIM, PSNR, and VMAF. The results show that DeepGame can achieve bandwidth savings of up to 36%, while maintaining the same perceived quality for players.

REFERENCES

[1] [n.d.]. Amazon Pushes into Making Video Games, Not Just Streaming Their Play. https://nyti.ms/2RwJMD5. [Online; accessed April 2021].

[2] 2021. DeepGame dataset: Network and Systems Lab, Simon Fraser University. https://nmsl.cs.sfu.ca/index.php/Resources#Datasets

[3] AMD. [n.d.]. Advanced Media Framework SDK. https://bit.ly/3tiASXK. [Online; accessed April 2021].

[4] AMD. [n.d.]. Google Partners with AMD for Custom Stadia GPU. https://bit.ly/3g4RheL. [Online; accessed April 2021].

[5] Junjie Bai, Fang Lu, Ke Zhang, et al. 2019. ONNX: Open Neural Network Exchange. https://github.com/onnx/onnx.

[6] Ankita Bhutani and Preeti Wadhwani. 2019. Cloud Gaming Market Size By Type, By Device, By Business Model, Industry Analysis Report, Regional Outlook, Growth Potential, Competitive Market Share and Forecast, 2019– 2025. https://www.gminsights.com/industry-analysis/cloud-gaming-market

[7] K. Bilal and A. Erbad. 2017. Impact of Multiple Video Representations in Live Streaming: A Cost, Bandwidth, and QoE Analysis. In *2017 IEEE International Conference on Cloud Engineering (IC2E)*. 88–94. https://doi.org/10.1109/IC2E.2017.20

[8] G. Bjontegaard. 2001. Calculation of Average PSNR Differences between RD-curves.

[9] Yue Chen, Debargha Murherjee, Jingning Han, Adrian Grange, Yaowu Xu, Zoe Liu, Sarah Parker, Cheng Chen, Hui Su, Urvang Joshi, Ching-Han Chiang, Yunqing Wang, Paul Wilkins, Jim Bankoski, Luc Trudeau, Nathan Egge, Jean-Marc Valin, Thomas Davies, Steinar Midtskogen, Andrey Norkin, and Peter de Rivaz. 2018. An Overview of Core Coding Tools in the AV1 Video Codec. In *2018 Picture Coding Symposium (PCS)*. IEEE. https://doi.org/10.1109/pcs.2018.8456249

[10] Sharon Choy, Bernard Wong, Gwendal Simon, and Catherine Rosenberg. 2012. The Brewing Storm in Cloud Gaming: A Measurement Study on Cloud to End-User Latency. In *Proc. of NetGames'12*. Venice, Italy, Article 2, 6 pages.

[11] Mark Claypool and Kajal Claypool. 2010. Latency Can Kill: Precision and Deadline in Online Games. In *Proc of ACM MMSys'10*. Phoenix, AZ, 215–222.

[12] Michael Dorr, Thomas Martinetz, Karl R. Gegenfurtner, and Erhardt Barth. 2010. Variability of eye movements when viewing dynamic natural scenes. *Journal of Vision* 10, 10 (Aug. 2010), 28–28.

[13] Sadjad Fouladi, John Emmons, Emre Orbay, Catherine Wu, Riad S. Wahby, and Keith Winstein. 2018. Salsify: Low-Latency Network Video through Tighter Integration between a Video Codec and a Transport Protocol. In *Proc. of USENIX NSDI'18)*. Renton, WA, 267–282.

[14] Google. [n.d.]. Google Stadia. https://stadia.google.com/. [Online; accessed April 2021].

[15] Google. [n.d.]. Stadia Help: Bandwidth, data usage, and stream quality. https://bit.ly/37sfw13. [Online; accessed February 2021].

[16] Félix G. Harvey, Mike Yurick, Derek Nowrouzezahrai, and Christopher Pal. 2020. Robust motion in-betweening. *ACM Transactions on Graphics* 39, 4 (jul 2020). https://doi.org/10.1145/3386569.3392480

[17] Mohamed Hegazy, Khaled Diab, Mehdi Saeedi, Boris Ivanovic, Ihab Amer, Yang Liu, Gabor Sines, and Mohamed Hefeeda. 2019. Content-aware video encoding for cloud gaming. In *Proceedings of the 10th ACM Multimedia Systems Conference*. ACM. https://doi.org/10.1145/3304109.3306222

[18] John M. Henderson, James R. Brockmole, Monica S. Castelhano, and Michael Mack. 2007. Chapter 25 - Visual saliency does not account for eye movements during visual search in real-world scenes. In *Eye Movements*. Elsevier, 537–III.

[19] Chun-Ying Huang, Kuan-Ta Chen, De-Yu Chen, Hwai-Jung Hsu, and Cheng-Hsin Hsu. 2014. GamingAnywhere: The First Open Source Cloud Gaming System. *ACM Trans. Multimedia Comput. Commun. Appl.* 10, 1s, Article 10 (Jan. 2014), 25 pages. https://doi.org/10.1145/2537855

[20] Gazi Karam Illahi, Thomas Van Gemert, Matti Siekkinen, Enrico Masala, Antti Oulasvirta, and Antti Ylä-Jääski. 2020. Cloud Gaming with Foveated Video Encoding. *ACM Trans. Multimedia Comput. Commun. Appl.* 16, 1, Article 7 (Feb. 2020), 24 pages. https://doi.org/10.1145/3369110

[21] Intel. [n.d.]. Intel Quick Sync Video. https://intel.ly/2NMlFys. [Online; accessed February 2021].

[22] ITU-T. 2008. P.910 : Subjective video quality assessment methods for multimedia applications. https://www.itu.int/rec/T-REC-P.910-200804-I

[23] Anton S. Kaplanyan, Anton Sochenov, Thomas Leimkühler, Mikhail Okunev, Todd Goodall, and Gizem Rufo. 2019. DeepFovea. *ACM Transactions on Graphics* 38, 6 (nov 2019), 1–13. https://doi.org/10.1145/3355089.3356557

[24] B. Li, H. Li, L. Li, and J. Zhang. 2014. λ Domain Rate Control Algorithm for High Efficiency Video Coding. *IEEE Transactions on Image Processing* 23, 9 (2014), 3841–3854. https://doi.org/10.1109/TIP.2014.2336550

[25] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. 2018. Focal Loss for Dense Object Detection. arXiv:1708.02002 [cs.CV]

[26] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. 2014. Microsoft COCO: Common Objects in Context. In *Computer Vision – ECCV 2014*. Springer International Publishing, 740–755. https://doi.org/10.1007/978-3-319-10602-1_48

[27] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. 2016. SSD: Single Shot MultiBox Detector. *Lecture Notes in Computer Science* (2016), 21–37. https://doi.org/10.1007/978-3-319-46448-0_2

[28] Comics Gaming Magazine. [n.d.]. google stadia hits 1 millionusers. https://www.cgmagonline.com/2020/04/23/google-stadia-hits-1-million-users/. [Online; accessed April 2021].

[29] Shie Mannor, Dori Peleg, and Reuven Rubinstein. 2005. The Cross Entropy Method for Classification. In *Proceedings of the 22nd International Conference on Machine Learning* (Bonn, Germany) *(ICML '05)*. Association for Computing Machinery, New York, NY, USA, 561–568. https://doi.org/10.1145/1102351.1102422

[30] Microsoft. [n.d.]. Microsoft Xbox Cloud Gaming. https://bit.ly/3wLW7U9. [Online; accessed April 2021].

[31] Debargha Mukherjee, Jingning Han, Jim Bankoski, Ronald Bultje, Adrian Grange, John Koleszar, Paul Wilkins, and Yaowu Xu. 2013. A Technical Overview of VP9 – The Latest Open-Source Video Codec. In *SMPTE 2013 Annual Technical Conference & Exhibition*. IEEE. https://doi.org/10.5594/m001518

[32] Nvidia. [n.d.]. Nvidia GeForce Now. https://bit.ly/32cIOhn. [Online; accessed April 2021].

[33] Nvidia. [n.d.]. NVIDIA Video Codec SDK. https://bit.ly/2ZxNo8I. [Online; accessed April 2021].

[34] Playstation. [n.d.]. PlayStation Now. https://bit.ly/2QmCtgp. [Online; accessed April 2021].

[35] Joseph Redmon. 2013–2016. Darknet: Open Source Neural Networks in C. http://pjreddie.com/darknet/.

[36] Joseph Redmon and Ali Farhadi. 2018. YOLOv3: An Incremental Improvement. *arXiv* (2018).

[37] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2017. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39, 6 (jun 2017), 1137–1149. https://doi.org/10.1109/tpami.2016.2577031

[38] G. J. Sullivan, J. Ohm, W. Han, and T. Wiegand. 2012. Overview of the High Efficiency Video Coding (HEVC) Standard. *IEEE Transactions on Circuits and Systems for Video Technology* 22, 12 (2012), 1649–1668.

[39] Gary J. Sullivan and Jens-Rainer Ohm. 2010. Recent developments in standardization of high efficiency video coding (HEVC). In *Applications of Digital Image Processing XXXIII*, Andrew G. Tescher (Ed.). SPIE. https://doi.org/10.1117/12.863486

[40] Wowza Media Systems. [n.d.]. What Is Low Latency and Who Needs It? https://bit.ly/2NkTNkP. [Online; accessed February 2021].

[41] Tobii. [n.d.]. Types of eye movement. https://bit.ly/3pANiaW. [Online; accessed February 2021].

[42] Tom Tullis and Bill Albert. 2013. Chapter 7 - Behavioral and Physiological Metrics. In *Measuring the User Experience (Second Edition)* (second edition ed.), Tom Tullis and Bill Albert (Eds.). Morgan Kaufmann, Boston, 163–186. https://doi.org/10.1016/B978-0-12-415781-1.00007-8

[43] Zhou Wang and A.C. Bovik. 2001. Embedded foveation image coding. *IEEE Transactions on Image Processing* 10, 10 (2001), 1397–1410. https://doi.org/10.1109/83.951527

[44] Jiangtao Wen, Meiyuan Fang, Minhao Tang, and Kuang Wu. 2015. R-(lambda) Model Based Improved Rate Control for HEVC with Pre-Encoding. In *2015 Data Compression Conference*. IEEE. https://doi.org/10.1109/dcc.2015.35

[45] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra. 2003. Overview of the H.264/AVC video coding standard. *IEEE Transactions on Circuits and Systems for Video Technology* 13, 7 (2003), 560–576.

[46] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. 2015. A Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP. In *Proc. of ACM SIGCOMM'15*. London, United Kingdom, 325–338.