Department of Computer Science

Submitted in part fulfilment for the degree of BEng

A Quantitative Study of the Extent to Which Region-of-Interest Can Be Used for Viewpoint Motion Prediction

In Industrial Partnership with

Joseph Silva

28th April 2025

Supervisor: Dr. William Smith and Thomas Oliver

Acknowledgements

I would like to thank my supervisors William Smith and Thomas Oliver for their great insights, guidance, and help throughout the project. The direction and quality of this work has been greatly improved through their input.

Contents

List of Figures

Executive Summary

This project aims to develop a predictive model utilising frame data to predict viewpoint motion in video games to enable pre-rendering and delivery of frames in a cloud gaming scenario. A further aim is to use the predictive model to reduce encode time by indicating which pixels are likely to remain in the frame.

The pre-rendering and delivery of frames have been used in Outatime [1] and was able to mask up to 120ms round trip time (RTT). The predictive model used in Outatime only utilised previous motion, and was implemented as a Markov Decision Process. If a predictive model that also utilises frame data can mask a longer RTT, it would enable a boost to the responsiveness of cloud gaming systems, which is one of the biggest blockers of the industry.

Two predictive models were developed, one using image data and one without to predict viewpoint motion. These were trained on a dataset of speedruns of the videogame Doom. Additionally, a method to use the viewpoint motion predictions to assist the video encoder was developed.

The predictive models were compared using Mean Squared Error and Percentage Error, and found that the image model was better than the motion only model. The success of the model indicated that a greater latency could be masked of up to 171ms, a 51ms increase over current methods.

Frames of the dataset were encoded with assistance of the viewpoint motion predictions from the models. The image model was able to assist the encoder to improve the frame quality according to VMAF, while being encoded at a 5kbps bitrate less than the regular encode.

The results show that image models have a huge potential to reduce latency in cloud gaming systems over current methods. They also demonstrate that predictive models in cloud gaming can also be used to lower data transmission as an added benefit of implementing predictive rendering.

Statement of Ethics

Throughout this paper, gameplay data from speed runs of the video game Doom is used. This data was collected as part of the ResearchDoom project, and is known as the CocoDoom dataset.

This dataset is released under a BDS-like license which can be found here.

The potential uses of this research have been considered, and the conclusion that it only poses positive benefits has been reached. The research aims to improve the user experience of cloud gaming systems, while also reducing data transmitted.

1 Introduction

1.1 Context

Client-server latency poses a massive problem in cloud gaming systems. Users expect responsiveness and low latency comparable to gaming locally, but most of this latency is due to the physical distance to a server and weak last-mile connections, making it extremely hard to reduce.

The latency in a cloud gaming system is made up of several components: processing, frame capture, encoding, transmission, decoding, rendering, and display. Most of these stages are optimized and are fixed contributors of latency. Transmission latency is variable and can depend on the household's network speeds and physical proximity to a cloud gaming server, while also being one of the biggest contributors of latency. Although improvements in network infrastructure can reduce transmission latency, this can take a while to affect the majority of households.

1.2 Motivation

Predictive or speculative based cloud gaming has been proposed and explored as a way to reduce latency [1], [2]. User input is predicted ahead of time, meaning that frames can be pre-rendered and delivered to the client so that it can respond immediately to the user input.

Outatime [1] implements a system to do this, by building a predictive model and recovering from mis-predictions, enabling the masking of up to 120ms latency. The predictive model that Outatime uses takes the last user input and uses a Markov Decision Process to predict the next movement. This ignores another potential input source, the visual frame data, which has proven to be helpful in predicting viewpoint when streaming 360-degree video [3].

Utilising frame data in addition to motion data to predict viewpoint motion in games has not been explored before. If a predictive model that takes

image data as input can predict more accurately further into the future, a larger RTT can be masked, enabling significant boosts to the latency of cloud gaming systems.

Another method of reducing latency is to provide assistance to the video encoder by indicating which regions of the frame should be compressed more/less. With the prediction of the upcoming viewpoint motion, this provides enough information to determine which pixels are likely to remain in the frame, and this can be used to indicate which regions should be compressed less. It is then possible that making a viewpoint motion prediction can both enable the pre-rendering and delivery of frames, while enabling frames to be compressed to a lower bitrate while maintaining the same Quality of Experience (QoE) for the user.

1.3 Objectives

The objective for this project is to enable further reduction in latency in cloud gaming systems by developing a predictive model using image data to predict player movement that can be used to pre-render and deliver frames, and to develop a method that uses these predictions to assist the video encoder.

This can be split into the following aims:

1. Pre-processing and formatting of the dataset

2. Develop and train a baseline and an image model to predict viewpoint movement, and compare them.

3. Develop a method that can assist the video encoder using viewpoint movement predictions, and assess it with the different predictive models using video encoding metrics.

2 Background

2.1 Cloud Gaming

In recent years, cloud gaming has proven a popular alternative to local execution of games, with 395 Million current users of cloud gaming (as of 2024), and this is predicted to grow to 493 Million in 2027 [4].

The general architecture of a cloud gaming system consists of a thin client that sends user input, and receives and displays rendered frames - and a cloud gaming server, that receives the user input and computes and renders the next frame.

This method has many advantages over games being locally rendered: devices can play games that they cannot render themselves, games are updated on the cloud server instead of on many client devices which avoids frustrating updates for end-users, and user progression data is seamlessly synced across multiple client devices.

However, cloud gaming also introduces potentially large latency, with round trip times (RTTs) that can exceed 100ms, a latency response that when found in multiplayer games can hinder gameplay, disrupt fluidity, and potentially cause "rubber-banding" [5]. The minimum expected frame rate for many gamers is 30 frames per second (FPS), which corresponds to a cloud RTT of 33ms.

This latency is made up of many different components required for cloud gaming, such as the transmission time between the server and the client, time to encode the frames, and client-side rendering/upscaling. How each of these fields contributes to the latency in an example cloud gaming system is shown below.

Of this added latency, the main contributor is the transmission latency. There are strategies to reduce this. Firstly, a heatmap indicating regions of interest in the frame can be provided to the video encoder, enabling a frame to be encoded at a lower bitrate, without loss of quality. This can reduce transmission latency by reducing the size of the frames to be transmitted.

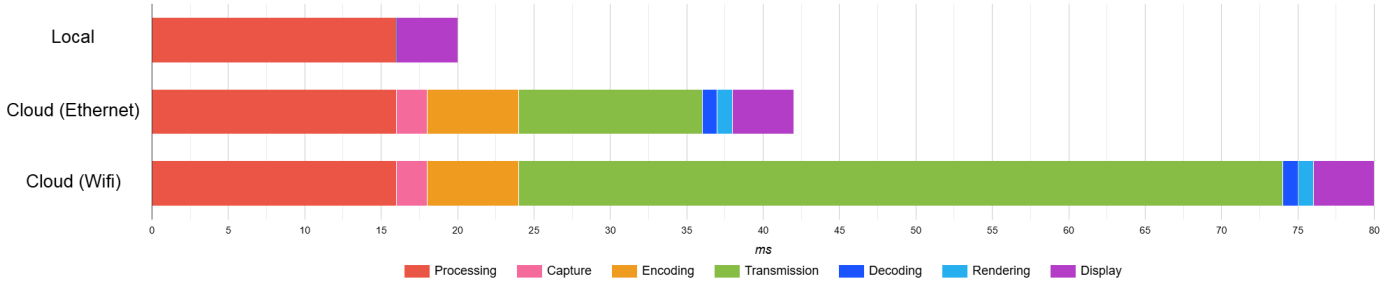


Figure 2.1: Stacked bar chart of the latency in a real cloud gaming system - data used from Shadow [6]

Another potential solution could be to build more data centres, which can then be physically located closer to a user. However, this would be extremely costly and doesn't account for the vast majority of network latency coming from last-mile connections (often about 10-40ms) [7].

Another solution is to compress the frames more, or to transmit the frame at a lower quality and upscale using DLSS or FSR on the client-side. This has the down-side of worse quality for the client however, and can only reduce the transmission latency by a certain amount. Even though less data is being transmitted, the travel time between the client and server still exists.

A way to get around this is to predict future user input on the server and pre-render and deliver the frames to the client. This saves a whole RTT, as the server does not have to wait to receive the user's next input before rendering the next frame. This method could even achieve 0 latency total, given perfect prediction far enough into the future, being able to offset more than just the transmission latency by starting the whole processing for a new frame much earlier than before.

In reality, perfect prediction is impossible, due to the random nature of player behaviour, however, there are many strategies to overcome this, such as delivering a wider Field-of-View (FoV) than needed, from which the correct region can be rendered on the client side, or applying Image Based Rendering on the client-side to recover from misprediction as applied in Lee [1].

2.2 360-Degree Video

Streaming 360-degree video poses similar challenges to cloud gaming, as streaming all areas of the video (even those not looked at by the user), would consume massive amounts of bandwidth, and ultimately most of

the streamed data would not be viewed. This has led to the development of many predictive methods to reduce the amount of bandwidth, which have proven to be extremely effective at reducing bandwidth usage while maintaining a high Quality of Experience (QoE) [8]–[10].

These solutions benefit from the fact that the 360-degree videos are predetermined, meaning that a predicted viewed section of a frame far in the future can be delivered to the client - and as long as the viewpoint prediction is accurate, it would be exactly correct, leading to no drop in QoE for the user.

Ultimately, when a model is unsure of the future movement of the user, or it is predicting the viewpoint for a frame far in the future, the server can simply extend the size of the streamed viewpoint, even up to the whole 360-degree frame. This caps the maximum bandwidth usage and, while streaming the whole 360 degrees would not be efficient, allows for a fallback in case of uncertainty in the prediction. This is not feasible in cloud gaming, where the state space quickly explodes due to being able to move through and interact with aspects of the world, instead of viewing it from one point.

When applying similar models to cloud gaming, we have to be able to predict more parameters of the viewpoint changing to account for the user moving through the space as well as looking around. We also have additional processing to do on the server, necessitating quicker prediction, and we are unable to deliver frames as far into the future as 360-degree video, due to them not being predetermined.

However, the major development of predictive models for viewpoint motion, due to the popularity of 360-degree video, becomes an extremely helpful reference point when considering research in viewpoint motion prediction.

2.3 Predictive Models

Here, models that are commonly used for viewpoint prediction are outlined. Most of these models have been developed for the viewpoint prediction in 360-degree video.

2.3.1 Recurrent Neural Networks

Recurrent neural networks process a sequence of vectors by applying a recurrence - using the previous state to define the current state. This is

implemented by having the previous state as part of the input to the neural network, which then predicts the next state.

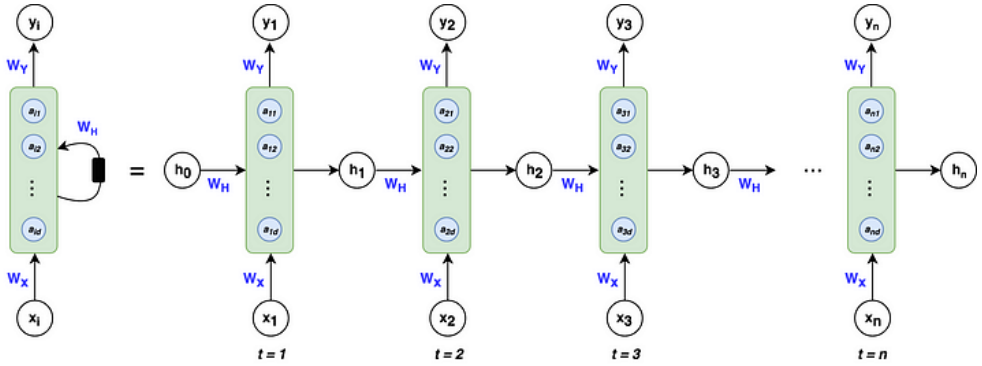


Figure 2.2: A many-to-many RNN, Khuong [11]

This can be trained using backpropagation in the same way as neural networks, although this can be truncated to be more memory-efficient.

The state is implanted as a hidden vector h_t. This can cause a loss of important information as information about the entire sequence becomes embedded in a fixed-size vector.

The recurrent neural network architecture is the basis for Long Short-Term Memory, which is the model opted for by most viewpoint prediction methods [12], [13]. It can also be seen as a helpful precursor to understanding transformers.

2.3.2 Long Short-Term Memory

Long Short-Term Memory (LSTM) is a type of recurrent network architecture that is designed to overcome blown-up or vanishing error, which can lead to oscillating weights or difficulties bridging long time lags [14].

This is implemented using memory cells and gate units - input, output, and forget gates. The forget gate controls how much of the long-term memory to remember, the input gate controls what to add to the long-term memory, and the output gate controls what should be output from the specific node using information from both the previous state and the long-term memory.

This model proves to be extremely powerful for predicting from sequential data. This architecture is commonly used in viewpoint prediction, as using sequential data improves the prediction accuracy and the LSTM model improves on recurrent neural networks.

Commonly this model is used to predict from the previous viewpoint position

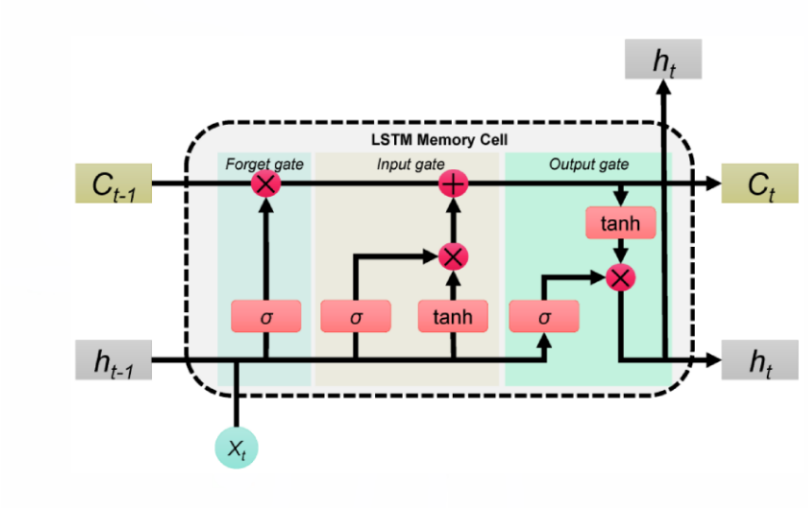


Figure 2.3: A memory cell in a LSTM, Yehia [15]

and location, without any context of the actual viewed content. This can lead to a large error in the yaw prediction, as the viewed content can quickly make the yaw movement erratic.

2.4 Region-of-Interest Data

2.4.1 The effect of ROI Data in 360-degree videos

In LSTMs, when the Region-of-Interest data is combined with viewpoint movement in 360-degree videos, this has been found to decrease performance compared to predicting from the viewpoint movement alone [3]. However, it was also found that when predicting for periods longer than 2-3 seconds, including the Region-of-Interest data in the prediction can reduce error. We can see fundamentally why this may be true, with very short predictions being mainly defined by the user's small head movements and tendencies, and longer predictions being better defined by Regions-of-Interest.

Another finding of Rondon [3] is that the type of content being viewed can have a significant effect on the impact of Region-of-Interest data. For example, "Exploration" type videos have a worse viewed content prediction accuracy than "Non-Exploration" type videos, which often have a more clearly defined Region-of-Interest.

A key note here is that video games have both types of movement. Often,

when first loaded or entering a new room, the player will look around in an exploratory way, whereas when in, for example, combat, Regions-of-Interest have a key role in the player's movement.

2.4.2 How should Regions-of-Interest be represented?

Saliency Maps

Saliency maps are a visual representation of saliency (how much something stands out). These are often used to visualize the interaction/attention of a participant within regions of an image. Saliency maps have been used in many fields, including UI design [16], foveation for compression [17] and foveation for rendering [18].

Saliency maps can be derived in many ways including: through static methods (such as Itti and Koch [19]), from motion in video frames [20], or through a deep learning based method (e.g. TASED-Net [21], STRA-Net [22]).

Human-generated saliency maps have been found to help with generalization of deep learning for classification on a single image [23]. Having access to human-generated saliency maps for a specific frame could then potentially be helpful in other prediction tasks, however, to get this for a specific frame in our scenario would likely require specific hardware, such as an eye-tracker.

Viewpoint prediction models for 360 degree video have been trained using saliency as in Xu [24], where a saliency map is generated using SalNet [25], features are then extracted with a CNN, while the previous movement is encoded by an LSTM. These features are then combined to make the movement prediction. They find that this method improves accuracy over the RGB image alone.

Using a saliency map would also enable the use of foveation for compression, which would enable greater compression of the frame, resulting in a smaller RTT. This can also be used for foveated encoding, and can reduce the time taken to encode a frame before transmission, reducing the latency time.

Semantic Segmentation

Semantic segmentation assigns a label to each pixel in an image. This can be represented as a segmentation map. A segmentation map is much simpler than the original image, and has been shown to improve reinforcement learning in video game environments [26].

This makes it easier to train well across visually different levels. It is reasonable to assume that this boost will transfer into supervised learning as well, due to the simplification of the game data, and the diverse range of environments commonly found in video games.

Segmentation maps can be computed through a simple shader, which can be computed extremely quickly. This adds minimal overhead to the game's performance. An example implementation of this is by Bubeníček [27].

2.5 Encoding Metrics

There exist many metrics to measure the performance of video encoding, here we use 2 common metrics: Peak Signal-to-Noise Ratio (PSNR) and Video Multimethod Assessment Fusion (VMAF).

2.5.1 PSNR

PSNR directly compares an altered image with it's original version. This is done through mean squared error (MSE) between the images, and then expressed as a logarithmic ratio relative to the maximum possible pixel value. When comparing a vanilla encode with an encode using ROI techniques, PSNR provides a measure of how the quality of the frame has changed. This helps to determine if the ROI approach maintains acceptable quality in the frame while optimizing some areas for viewer interest.

2.5.2 VMAF

VMAF is a metric developed by Netflix that predicts subjective video quality [28]. It uses a Support Vector Machine regressor to predict subjective human perception of video quality from multiple elementary quality metrics. The metric is represented as a score between 0 and 100, with higher scores representing a perceived higher quality.

By running a vanilla encode and a ROI encode to the same bitrate, we can compare VMAF to predict which method is more likely to appear as a higher quality to a user. This metric is very helpful to assess the impact of our ROI selection on encoding - if we select the most relevant ROIs and these have the least compression, this should be reflected in an increase in VMAF.

2.6 Related Projects

2.6.1 Outatime

Outatime [1] is the most similar project to this paper. It performs viewpoint prediction for cloud gaming, and implements a full rendering and frame delivery system, with checkpointing and rollback that can be fully used on real games played on cloud gaming servers. This system is able to mask up to 120ms of network latency.

Outatime creates 2 models - one for novice players, and one for experienced players, as they found that as long as train and test players are of a similar level, the impact of having a different player than the model was trained on use the system is marginal.

The prediction method Outatime uses is a Markov Decision Process based only on the past viewpoint movement of the player. Outatime is trained on both Doom 3 and Fable 3.

2.6.2 STAR-VP

STAR-VP [9] is very similar to the project, as it combines viewpoint movement and image data. They combine both previous viewpoint movement and image data by fusing them in a spatially-aligned way, by combining 3D spatial coordinates with a saliency value for the image data.

As the relative importance of previous viewpoint movement data and image data varies across prediction time steps, they use a two-stage fusion approach to address the time-varying importance. Namely, a Temporal Attention Module that adjusts attention to the different data types, followed by a Gating Fusion Module that fuses the results of the short term prediction module and the long term prediction module.

This architecture addresses concerns about combining movement data and

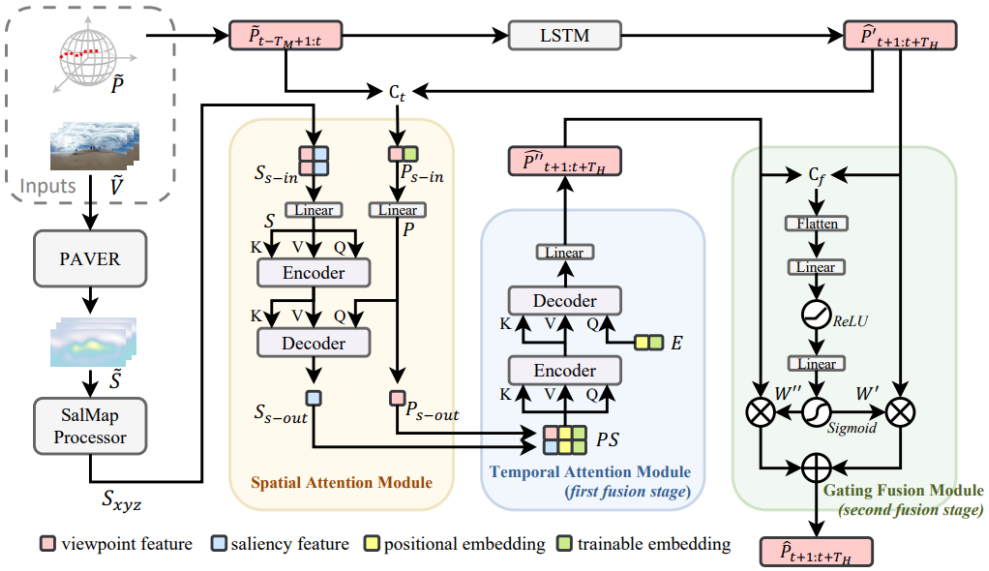


Figure 2.4: STAR-VP Model Overview [9]

image data that are relevant in game data, and so STAR-VP's solutions to these are extremely relevant to our problem.

3 Methodology

3.1 Overview

This section outlines the methodology used in training a multimodal network using previous motion vector and semantic segmentation information to predict future motion. The dataset and input representation are discussed and explained, and the architecture and training of the baseline model and multimodal modal are explored in depth.

3.2 Dataset

The CocoDoom dataset [29] is used. It is made up of pre-recorded frames from speedruns of the video game Doom. Each frame records the RGB frame and a wealth of other information, such as the player's position. The dataset also records the depth map at the frame, in which each pixel represents the distance between the camera and the first item, and the semantic segmentation map, in which each pixel represents what object is visible.

This dataset was opted for as it provides a wealth of data that is pre-computed. Also considered was extracting data from self-recorded gameplay, but this would provide no benefit over the data that is already provided in CocoDoom. Doom only allows for yaw rotation, there is no pitch rotation. In most models, the yaw rotation is the most erroneous, so this does not simplify the prediction much.

The data used as input to the model is the previous motion vector to the current frame, the semantic segmentation map, and the depth map. These are shown in Figure 3.1.

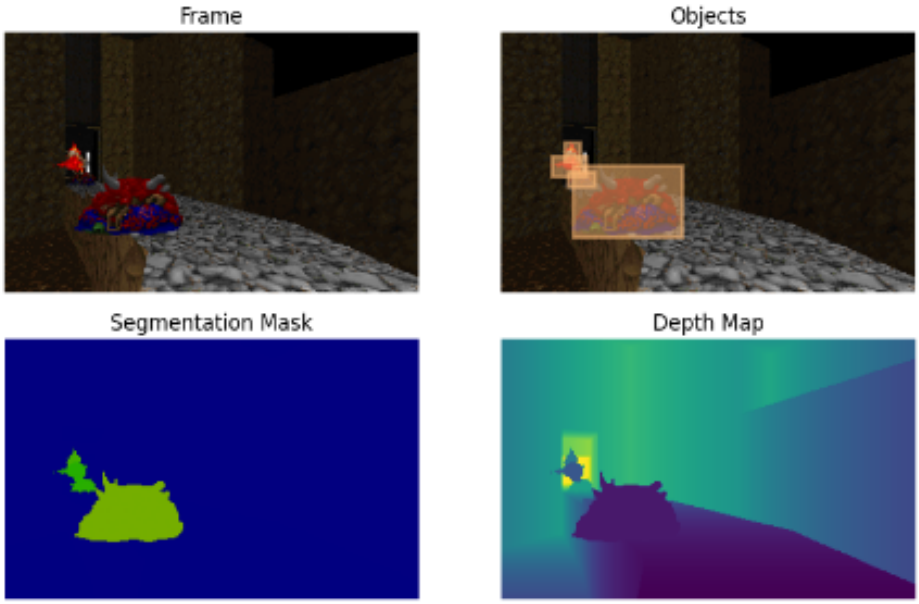


Figure 3.1: Segmentation mask and depth map from an example frame. Also included in the dataset is bounding boxes for each object, which is shown top right - while not used in the model, this is shown as an example of other data in the dataset.

3.2.1 Motion Vectors

Motion vectors represent a change in position between 2 time frames. The CocoDoom dataset provides the player position and yaw at every tic, so the motion vectors can easily be calculated by finding the difference between these positions.

One limitation of this representation is that as the player moves around the map, due to their yaw changing, the direction straight in front of them will vary between being primarily in the x axis, and primarily in the y axis. Given an input of the previous motion vector, we only know how much the yaw has changed from the previous frame - and so we have no knowledge of which direction the player is facing. This makes it very hard to predict how visual data will affect the player's movement - they could be facing the x-direction, or the y, or any angle - all with different impacts on the motion vector.

A solution to this is to represent the motion vector in a different way. Instead of having x and y, which are relative to the game map - we can define x' and y', which are relative to the viewing direction of the player.

If we take x' to be the amount moved in the direction parallel to the previous viewing direction of the player, and y' to be the amount moved in the

direction perpendicular to the previous viewing direction, we can represent the change in position in a way that is easy to relate to the visual data also input into our model.

This can be calculated as follows:

$$x' = xcos(2\pi - \theta) + ycos(\theta - \frac{1}{2}\pi) \tag{3.1}$$

$$y' = xsin(2\pi - \theta) + ysin(\theta - \frac{1}{2}\pi) \tag{3.2}$$

where x', y' are the transformed coordinates, x, y are the previous coordinates, and θ is the yaw.

3.2.2 Semantic Segmentation Representation

In CocoDoom, the semantic segmentation maps are encoded in a 24-bit RGB PNG image. For a pixel, the class identifier c can be calculated by combining the R, G, and B 8-bit values:

$$c = R + G \cdot 2^8 + B \cdot 2^{16} \tag{3.3}$$

The c value is coded to be able to represent the sky, a horizontal surface, or a vertical surface, or an object instance identifier. An object instance identifier is used to distinguish between different objects of the same class in CocoDoom. This identifies a unique instance of an object, and remains the same across frames, allowing a relation between frames to be established.

There are 8388608 object instance ids that could be used in CocoDoom - without a way to find the underlying class. For this reason, the only class information encoded was that of the sky, horizontal surface, vertical surface, or other.

Encoding these values to be passed into the network was done using one-hot encoding. For each pixel of the image, a one-hot vector indicating which class was present was used. A one-hot vector is commonly used to represent class information by representing each class as a different dimension, avoiding any implied relation between the classes.

3.2.3 Teleportation

Doom features teleportation - this results in very large motion vectors that are quite unpredictable. As the main focus of this paper is on the impact

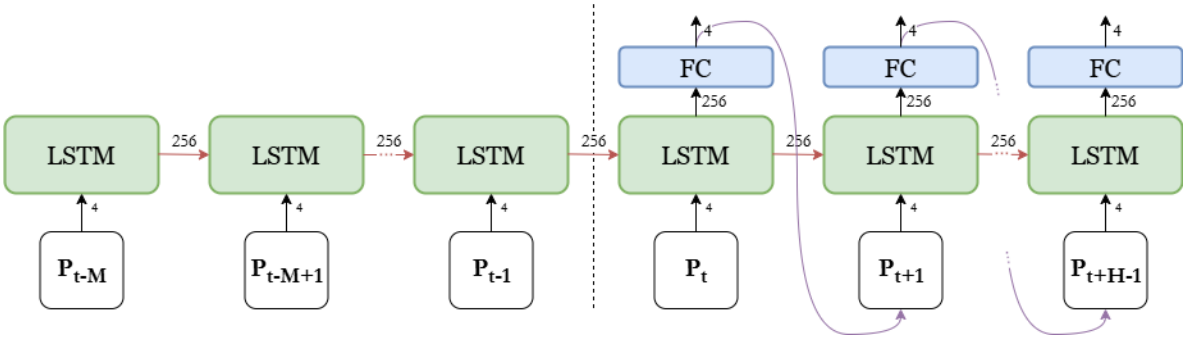


Figure 3.2: Seq2Seq Architecture for the baseline model. Left hand side represents the encoder and right hand side the decoder. P_t is the last input motion vector, M is the size of the input window, and H is the size of the prediction window.

of Region-of-Interest data on viewpoint motion prediction, teleportation instances were omitted from training. Predicting teleportation is more similar to predicting other impulse events - such as firing a weapon, or interacting with the environment. While there is room in future work to explore the prediction of these events, this is not explored here.

3.3 Baseline Model

A model trained only using previous motion vectors (as defined in eqs. (3.1) and (3.2)) was created as a baseline and for comparison, shown in Figure 3.2, where P_t is the motion vector at the point of prediction and M is the size of the input window, and H is the size of the prediction window. This means that the model takes in the M motion vectors previous to the frame, and outputs H motion vector predictions.

The model was based on LSTMs and uses a seq2seq architecture [30]. A seq2seq architecture consists of an encoder and a decoder. The M input motion vectors are input to the encoder, which generates and maintains a representation that is passed to the decoder, which predicts H motion vector predictions, which it does by making one prediction at a time that is fed back as input for the next prediction.

This architecture is also used in Rondon [3] as a baseline. This serves as a good comparison to the effect of using ROI data, as it utilises the context of previous frames.

Both the encoder and decoder consist of one LSTM layer with a hidden state with 256 features. For the decoder the LSTM layer is used in a loop

for each frame in the future to be predicted.

You can expect that this model will outperform a simpler model, just predicting the same input - as this model takes in a prediction window, it is able to infer non-linear movement. If the player is walking in a circle, predicting the same input would result in a prediction that walks in a straight line, whereas using this method with a prediction window could realise the trend in movement and keep predicting movement in a circle.

3.3.1 Training

The data split used was the run data split defined by CocoDoom. This split uses 1 run for training, 1 run for testing, and 1 run for validation. A run is one full playthrough of the game.

The baseline model was trained using a batch size of 256 over 20 epochs. Mean squared error was used as the criterion, and the Adam [31] optimizer was used. The Adam optimizer has an adaptive learning rate and requires little tuning of hyper-parameters and has been used widely in practice.

3.4 Multimodal Model

A limitation of the previous model is that if the user is directly looking at a wall (visible clearly using the depth map), and was walking forward previously, the model will likely predict that the user will keep moving forward, even though this is clearly not possible. If the user encounters an enemy (visible through the semantic segmentation map), this similarly will also affect their movement, but this cannot be represented by the previous model at all.

Training a model using image data aims to be able to encounter these situations in a more intelligent way, and result in more accurate predictions.

3.4.1 Inputs

The main model takes 3 types of input: previous motion vector, semantic segmentation map, and depth map. The dimensions are shown in the below table 3.1.

Table 3.1: Inputs and Dimensions

Input	Dimension
Motion Vector	1x4
Semantic Segmentation Map	200x320x4
Depth Mask	200x320

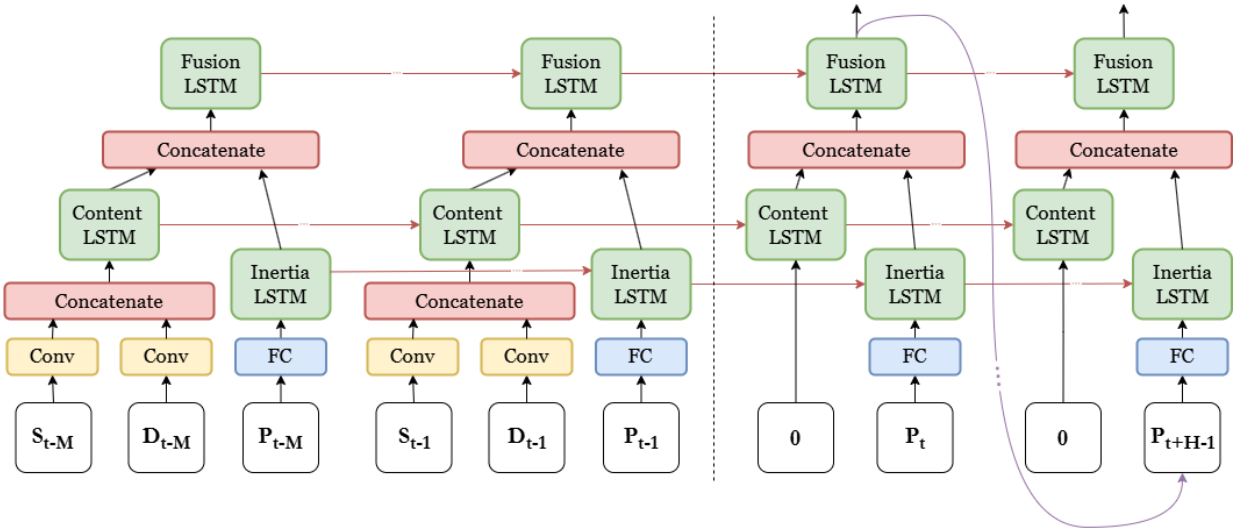


Figure 3.3: Seq2Seq Architecture for the image model. P, S, D are the motion vector, semantic segmentation map, and depth map. M is the size of the input window, and H is the size of the prediction window. The content LSTM in the decoder receives a masked input of 0.

3.4.2 Architecture

The seq2seq approach is taken again, with the encoder taking all 3 types of input, and the decoder making predictions using the hidden state, and the predicted motion vector for the frame before. This is represented in Figure 3.3, where S_t is the semantic segmentation map at the point of prediction and D_t is the depth map.

Encoder

Content LSTM - The semantic segmentation map and depth map are processed by convolution layers before being concatenated and passed into the content LSTM, which has a hidden size of 256. This LSTM maintains a hidden state representing the content of the frames in the input window.

Inertia LSTM - The previous motion data is processed by a fully connected layer before being passed to the inertia LSTM, with a hidden size of 256. This LSTM maintains a hidden state representing the movement of the player over the input window.

Fusion LSTM - The output of the content and inertia LSTMs are concatenated and input into the Fusion LSTM, which has a hidden size of 256. The goal of this LSTM is to fuse the outputs of the 2 LSTMs in a helpful way.

Decoder

The decoder consists of the same components as the encoder but with different weights. Each of the components receives the hidden state from the encoder of the corresponding component. Here, however, the visual LSTM receives a masked input reflecting the lack of knowledge of the next frame. The output of the decoder is re-injected as the motion vector input for the next prediction. This is shown in Figure 3.3.

A huge advantage of using the seq2seq architecture is for predicting multiple frames into the future. Due to variable RTT, a model that can predict an arbitrary number of frames into the future is desirable - this prevents developing a model that can only predict a fixed number of frames into the future.

To do this, a model that predicts 1 frame into the future is ideal, if we can re-inject the output of the model to predict for the next frame. A model like this is better than one that can predict for several frames into the future, as all training is focused on predicting 1 frame into the future well, whereas training a model to be able to predict a large range of frames would make it harder to train a specific frame prediction due to the training being spread.

Developing a model that can predict a further frame from re-injection of its output is hard in this case, as our input consists of not only the previous motion vector, but also the semantic segmentation map and depth map. A solution to this would be to train the model utilising masking - where we randomly hide the input (in this case the visual data) - and still train the model. This enables the model to predict well even when no visual data is available.

A cleaner approach is to use an encoder to encode the previous data in a hidden state, and a decoder that decodes this hidden state and predicts the next motion data. As the previous data is different from the predicted data, it makes sense to have different units to handle these.

3.4.3 Training

Training was performed using the Adam optimizer and mean squared error as the criterion, with a batch size of 256 over 20 epochs. A window method was used where the encoder receives the historic input, and the decoder predicts for a specified horizon.

3.5 Encoding

Viewpoint motion predictions, as output from the predictive models above, can be used to determine which pixels are likely to remain in a frame. This can be used to assist the video encoder by indicating which regions should be compressed less.

A method to derive a heatmap to assist the encoder utilising the motion vector predictions was developed. The depth map for a frame was used to generate a point cloud, representing each pixel visible in the frame in 3D coordinates.

$$u = \frac{(x - c_x) \cdot z}{f_x} \tag{3.4}$$

$$w = \frac{(y - c_y) \cdot z}{f_y} \tag{3.5}$$

Where u is the points world coordinates on the x-axis, w is the points world coordinates on the y-axis, c_x and c_y are the centre offset, and f_x and f_y are the focal lengths.

A transformation and rotation equivalent to the predicted motion vector was then applied to the point cloud, and the points were reprojected.

$$\begin{bmatrix} u' \\ w' \\ z' \end{bmatrix} = \left(\begin{bmatrix} u \\ w \\ z \end{bmatrix} - \begin{bmatrix} x \\ y \\ z \end{bmatrix} \right) \cdot \begin{bmatrix} cos(\theta) & 0 & -sin(\theta) \\ 0 & 1 & 0 \\ sin(\theta) & 0 & cos(\theta) \end{bmatrix} \tag{3.6}$$

Here x, y, z represent the 3D coordinates in the predicted motion vector, θ represents the angle of the motion vector. The 3x3 matrix represents a rotation around the y-axis. A binary heatmap is then generated representing each pixel in the original image with a 1 if the pixel is visible in the next frame, or a 0 if not visible. An example heatmap from a frame is shown in Figure 3.5.

By predicting motion vectors for the next n frames in the prediction window, n heatmaps can be combined. This was done by normalising each

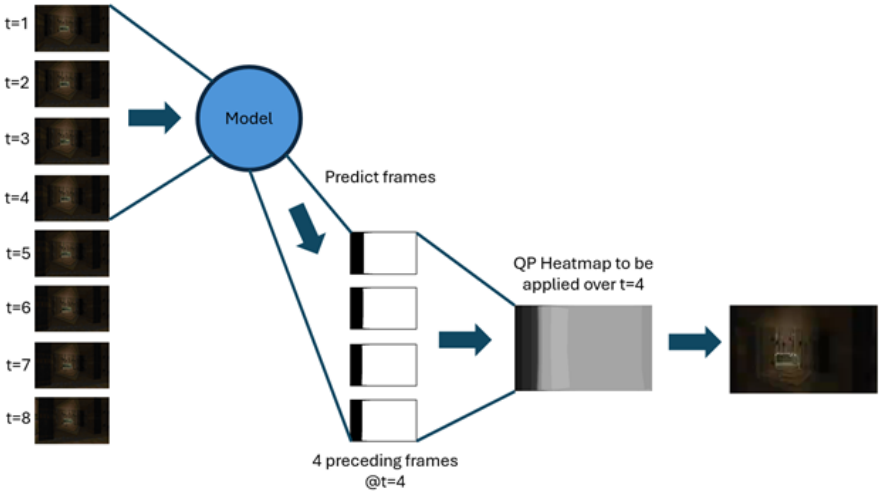


Figure 3.4: QP Heatmap Generation

heatmap (0.0-1.0), then applying a Gaussian blur. This was done to create natural gradients, preventing encoding artifacts. The heatmaps were then combined according to a weighting scheme: front-heavy, back-heavy, or linear. The front-heavy scheme prioritises contribution from the immediate future frames, taking less contribution from frames further in the future. Back-heavy prioritises distant predictions, while linear takes an equal contribution from each prediction. The linear method was found to be most appropriate for this context. Each scheme normalises contributions to sum to 1.0.

By passing these heatmaps to the encoder as QP maps, the encoder can dynamically adjust Quantization Parameters across regions of the frame. Smaller QP values are assigned to important regions, and larger QP values are assigned to less relevant regions. This helps the encoder to ensure important areas are at a higher quality, while less important areas can be compressed more [32].

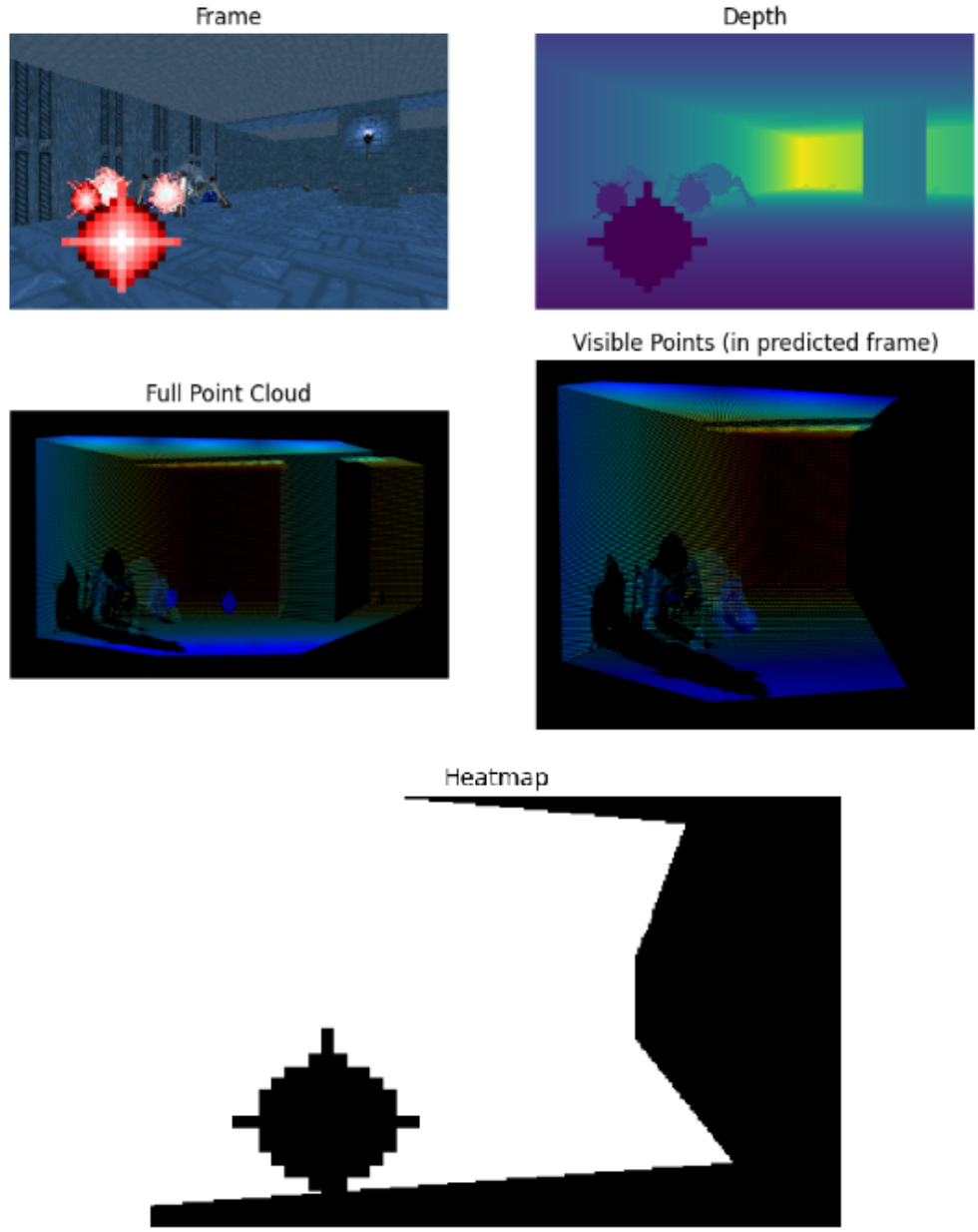


Figure 3.5: The transformed point cloud and generated heatmap for a motion vector of [13.56, 18.76, -4.07, 0.29]. Full point cloud shows the point cloud generated by projecting the depth map of the frame. Visible points represents the point cloud after removing points that are not visible after moving by the amount specified by the motion vector. The heatmap is white for pixels that remain in the frame, and black for pixels that do not.

4 Results and Evaluation

In this section, the predictive performance of the image model is measured and compared with the baseline model. Then the impact of the model on video encoding is assessed through a variety of metrics, and the suitability of the model in real-world applications is discussed.

4.0.1 MSE Loss

One of the primary ways of assessing the models performance is to compare the errors of the different models as the RTT increases (the models predict further into the future). For this, each of the models was assessed on the test dataset, predicting 10 frames (285ms) into the future.

As our approach for predicting 10 frames into the future also outputs the prediction for 9 frames, 8 frames, etc - we can then compute the error from the ground truth for a range of RTTs in one pass through the test dataset.

Figure 4.1 plots the MSE loss as the prediction window increases for the combined image and motion model, the previous motion-only model, and also a model that predicts the last movement as the next movement. We can see that both of the predictive models are much more accurate than predicting the last input, reducing the MSE loss from ~3750 to under 2000.

Plotting just the 2 predictive models, as done in Figure 4.2, shows interesting results. This was measured further for a predictive window of 20 frames. The image model shows an upwards trend - this makes sense, as the model predicts further into the future, its predictions will most likely get less accurate.

Interestingly, this does not stay the same for the motion model - its loss starts higher, then decreases before increasing at a slower rate than the loss of the image model, eventually beating the image model at the 7th frame (~200ms RTT).

These results show that the image model outperforms the motion only model for a short predictive window (0-200ms), whereas when predicting for a longer window (200-571ms)the motion only model performs best.

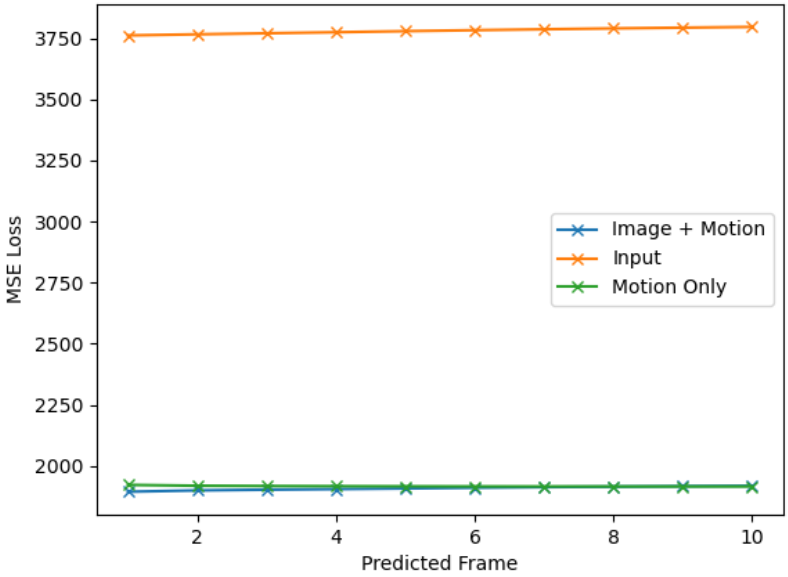


Figure 4.1: Loss for both predictive models and using prior input. The two predictive models (blue and green) clearly beat predicting the same last input (orange).

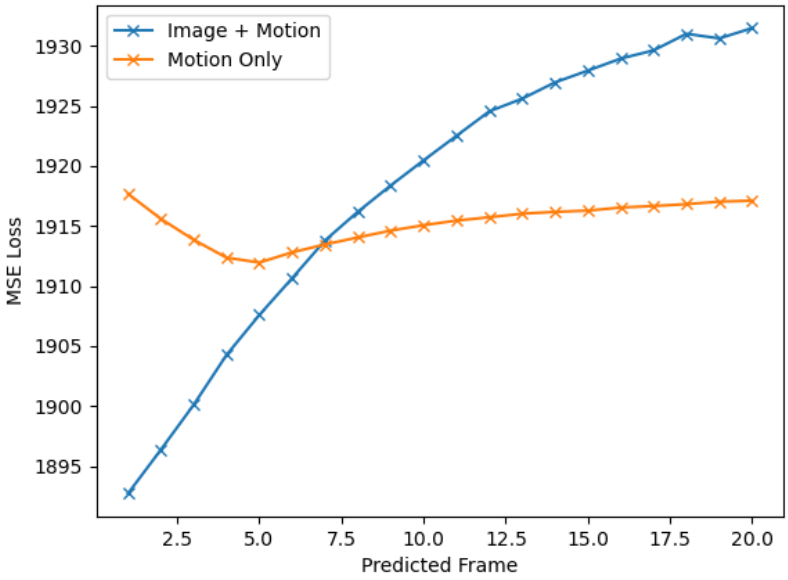


Figure 4.2: Image and baseline model loss. The image model (blue) beats the motion only model (orange) between frames 1-6 before being beaten by the motion model.

This result contradicts the findings of Rondon [3], which found motion only to be a better predictor for a short predictive window, and image data to be a better predictor for a long predictive window. This is likely due to the nature of the image model, as only a real frame, as displayed to the user, is input once to the image model - so as the image model predicts further, it doesn't receive any new information about the frames being rendered. This is different to in 360-degree video, where the whole video can be analysed, enabling changes in the image to help inform an image model.

However, as RTT in cloud gaming systems does not typically exceed 200ms, we can see that in a cloud gaming scenario, the predictions from the image model are more accurate.

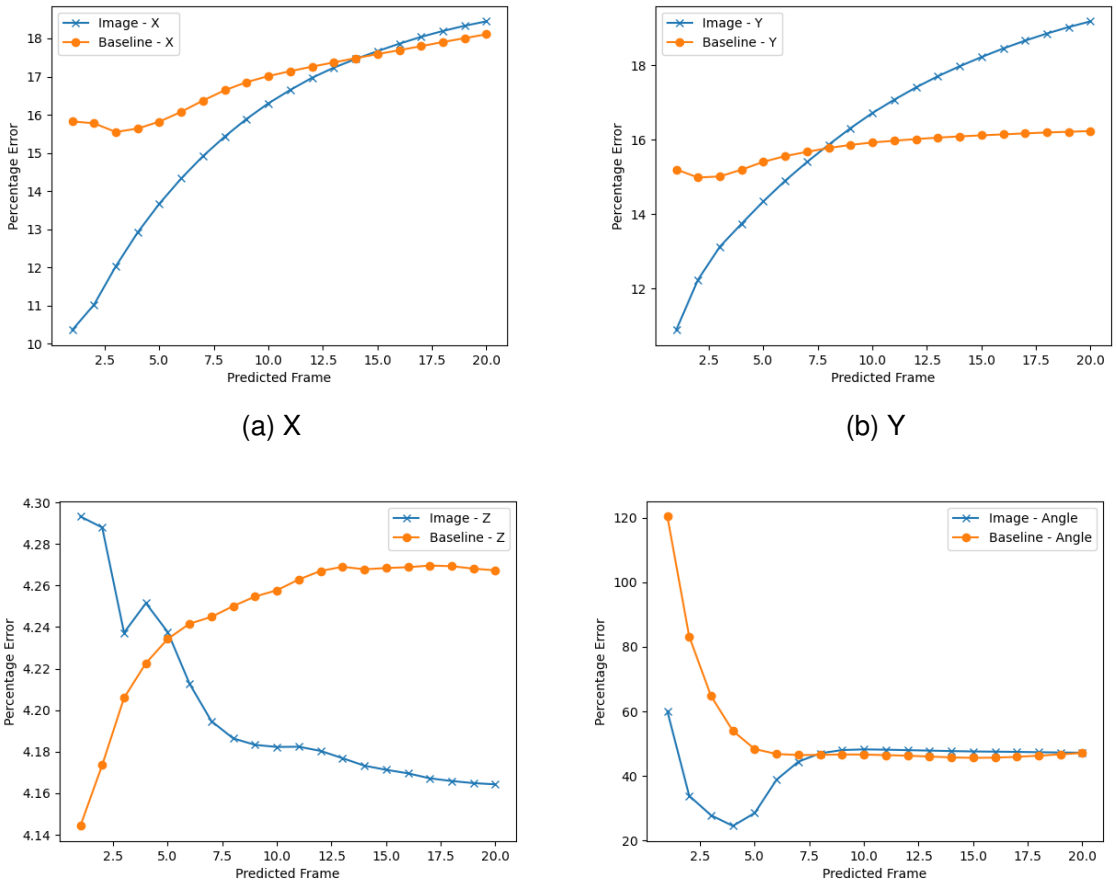
4.0.2 Percentage Error

The percentage error of each metric (X, Y, Z, Angle) for each frame predicted in a length 20 prediction window was calculated for the baseline and image model. The percentage error is based on how far the user could potentially move in a direction in 1 frame - if the prediction predicts a movement of 0, but the player moves as far as possible in 1 frame, it results in a percentage error of 100%.

These results are displayed in figs. 4.3a to 4.3d.

The X and Y metrics display similar trends, with the image model starting at a lower error than the baseline model, but increasing at a faster rate, with the baseline model beating the image model at frame 14 for the X metric, and at frame 8 for the Y metric. The percentage error for a RTT of 171ms (6 frame window) varies between 10-15% for both metrics in the image model, and between 15-16% for the baseline model. For these metrics, the image model is preferable for the RTT encountered in real-world cloud gaming systems.

The Z metric shows a different trend, where the baseline model beats the image model for predictions below 5 frames (143ms), with the image model then reducing error and beating the baseline model. However, the percentage error for both models is very low during the whole prediction window, only varying between 4.14-4.29%. The image model is likely better at predicting for longer periods due to being able to see changes in floor height, which could also worsen it's earlier predictions, where the nearby change in floor height isn't likely to have affected the player's position yet. This highlights a potential issue with reinjecting a previous prediction to the decoder - the decoder doesn't know how far away it is predicting from the initial input, which can lead to initial predictions paying too much attention



(a) X

(b) Y

(c) Z

(d) Angle

Figure 4.3: Percentage error for 20 frame prediction window. In all of the metrics apart from the Z metric, the image model (blue) beats the motion model in the short term (between around 1-6 frames). The Angle metric displays the highest percentage error, but is also the metric that the image model beats the motion model by the biggest margin.

to the encoded image data, which isn't likely to have affected the player's input yet.

The Angle metric shows very high percentage errors for both models, which then drops off and plateaus at ~50% for both models at around frame 7 (200ms). Before this, the image models comes in considerably lower than the baseline model, having a minimum value of 24.58%, compared to a minimum value of 45.64% for the baseline model. The inaccuracy in angle can be resolved by rendering a cube map, which can cover all 360 degrees. Predicting variance in addition to an angle prediction can be used to render a smaller portion of this cube map, called a clipped cube map. This method is utilised in Outatime [1], and they found that for their predictive model, 225° of coverage was suitable at 250ms RTT.

Overall, the image baseline performs better across the different metrics for predictions beyond the average RTT found in cloud gaming systems. From the percentage error alone, the image model clearly provides a benefit over the motion model for cloud gaming systems, with a lower percentage error for almost all metrics in a relevant prediction window. While the image model does not beat the motion model for the Z metric, it can be argued that the Z metric in Doom is not similar to that of other games - there is no jumping in Doom, meaning the Z metric changes only due to the cyclical walking pattern, and changes in map height. Using the same model on a game which does implement a jump mechanic might display different results - it enables the user to drastically change the Z value in the next frame with a single input.

4.0.3 Inference Time

The inference time of the 2 models was also measured, with the inference time of the image model being 24.25ms, and the baseline model being 6.57ms. This means that the baseline model has an advantage of 17.68ms over the image model. However we can see that the image model outperforms the baseline model even including this. Outatime [1] found that their predictive model (using motion only) could be used to mask up to 120ms RTT. Although they used a different predictive model, if we assume that when implemented in a real system our baseline model can mask a similar RTT, this corresponds to a prediction of about 4.25 frames into the future. The difference in inference time means that the image model would have to predict another 0.6 frames into the future, so we need to compare if the image model prediction at ~4.85 frames beats the baseline model at 4.25 frames. We can see that this is the case for MSE, and percentage error for all metrics apart from the Z metric. The image model at the 6th frame still remains below the MSE of the baseline model at 4.25 frames, indicating

that the image model could be used to potentially mask a RTT of 171ms, a reasonable increase from the 120ms previously achieved in Outatime.

4.0.4 Encoding Metrics

This section evaluates the image model's ability to inform the encoder, and compares it to the baseline model and a vanilla encode. 10 random frames were selected from the encoded dataset to analyse predictive and vanilla encoding against 3 codecs (H.264, H.265, AV1). The RDNA3 hardware platform was used.

The results are displayed in figs. 4.4a to 4.4f. Each row corresponds to one of the video codecs and has a separate graph for PSNR and VMAF.

PSNR

PSNR provides a measure of the quality of the encoded frame compared to the original frame. We can see that across the 3 codecs, the vanilla encode has the best PSNR (the highest), followed by the baseline encode, then the image encode.

This shows us that QP values are being set and it's affecting the encoding and so the final encoded frame. While this lowers the PSNR, this mainly just tells us that the frame is being changed as a result of our QP map. The image model was also encoded at a bitrate of 5kbps less than that of the vanilla and baseline model, which will have an impact.

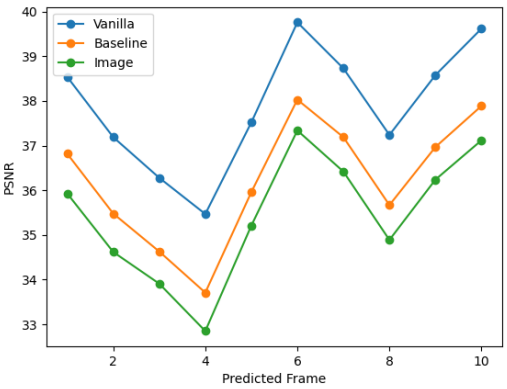
PSNR measures MSE across all pixels of the frame. As we are trying to indicate areas of interest so that a higher fidelity can be used in that area, while lowering fidelity in other areas, we need to use a better metric that predicts human perception.

VMAF

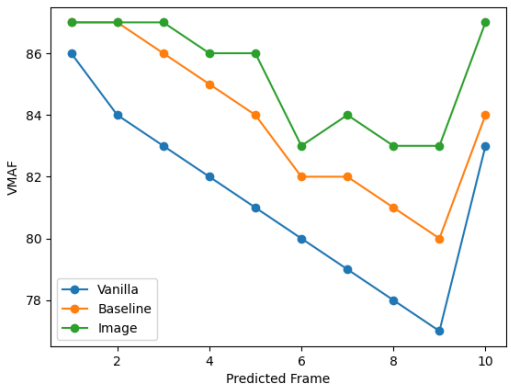
VMAF uses predicts subjective video quality by modeling human perception of video quality. This metric fits our use case and enables us to compare the VMAF between the different encoding methods. VMAF is represented as a score between 0 and 100. The results across the codecs are seen in the right-hand column of Figure 4.4.

These show that both the image and baseline model improve the quality of

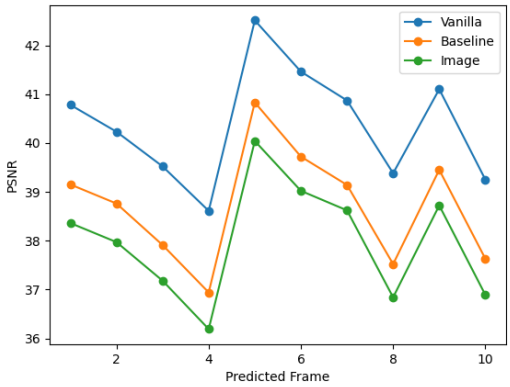
the frame over the vanilla encode. The image model beats both, which is a strong indicator of the success of assisting the encoder with heatmaps generated from the viewpoint movement predictions. This also indicates that the predictive model is predicting well and is fit for purpose.



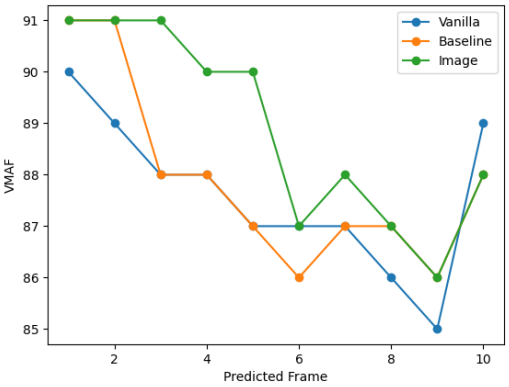
(a) H.264 PSNR



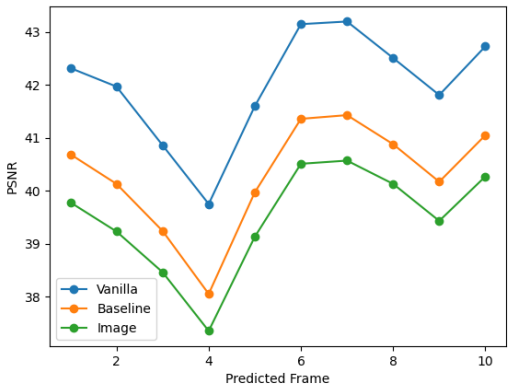
(b) H.264 VMAF



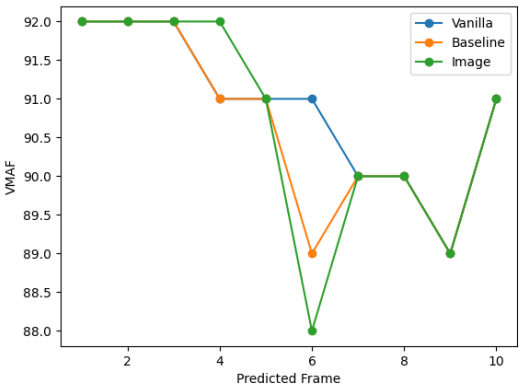
(c) H.265 PSNR



(d) H.265 VMAF



(e) AV1 PSNR



(f) AV1 VMAF

Figure 4.4: PSNR (left column) and VMAF (right column) for 3 video codecs (H.264, H.265, AV1) and 3 encodes (vanilla (blue), assisted by baseline model viewpoint predictions (orange), and assisted by image model viewpoint predictions (green)). The image model considerably beats the other 2 encodes in VMAF score, which measures the perceived quality of the frame.

5 Conclusion

5.0.1 Conclusion

A multimodal model has been successfully created and trained that takes in a combination of image and past movement data to predict viewpoint motion. This has demonstrably been shown to beat a motion-only approach across Mean Squared Error and Percentage Error. The performance of the model strongly suggests it could be used to mask a higher latency of up to 171ms in cloud gaming systems, representing a 51ms increase.

A method to use viewpoint predictions to assist the video encoder was developed, and the viewpoint predictions of the image model were found to improve the quality of the frames output, as measured through VMAF.

All 3 of the objectives were successfully met, and helped to solve a big problem within cloud gaming. The reduction of latency and the enabling of less data to be transmitted results in a more positive experience for cloud gaming users, and helps to remove some of the blockers that has affected the cloud gaming industry.

5.0.2 Future Work

The predictions of the image model were better than the predictions of the motion model at a short prediction window, which contradicts the findings of Rondon [3]. This can be attributed to the image model having no knowledge about what future frames will look like. This suggests that if the image model was updated with the result of future renders (the model predicts a movement, the frame is rendered and input back into the model for the next prediction), the performance of the model could be even better.

This would raise issues combining the inference time of the model with the rendering time of the frame, but this method could enable further latency reductions. To achieve this, the model would need to be implemented into a real speculative rendering system, so that real frames could be rendered using the viewpoint movement predictions.

Bibliography

[1] K. Lee, D. Chu, E. Cuervo *et al.*, 'Outatime: Using speculation to enable low-latency continuous interaction for mobile cloud gaming,' in *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, 2015, pp. 151–165.

[2] B. Anand and P. Wenren, 'Cloudhide: Towards latency hiding techniques for thin-client cloud gaming,' in *Proceedings of the on Thematic Workshops of ACM Multimedia 2017*, 2017, pp. 144–152.

[3] M. F. R. Rondon, L. Sassatelli, R. A. Pardo and F. Precioso, 'Revisiting deep architectures for head motion prediction in 360 $\{\backslash$deg$\}$ videos,' *arXiv preprint arXiv:1911.11702*, 2019.

[4] Statista, *Global cloud gaming users from 2017 to 2027 (in millions) [graph]*, Accessed: 2024-01-03, 2024. [Online]. Available: https://www.statista.com/forecasts/1390157/cloud-gaming-users-global.

[5] B. Place, *Ping latency in gaming*, Accessed: 2024-12-06, 2023. [Online]. Available: https://www.bandwidthplace.com/article/ping-latency-in-gaming.

[6] Shadow, *Roadmap: Cloud gaming without latency*, Accessed: 2025-02-20, 2025. [Online]. Available: https://shadow.tech/en-GB/blog/roadmap-cloud-gaming-without-latency.

[7] S. Sundaresan, W. de Donato, N. Feamster, R. Teixeira, S. Crawford and A. Pescapè, 'Broadband internet performance: A view from the gateway,' in *Proceedings of the ACM SIGCOMM 2011 Conference*, ser. SIGCOMM '11, Toronto, Ontario, Canada: Association for Computing Machinery, 2011, pp. 134–145, ISBN: 9781450307970. DOI: 10.1145/2018436.2018452. [Online]. Available: https://doi.org/10.1145/2018436.2018452.

[8] 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 *2016 IEEE International Conference on Big Data (Big Data)*, 2016, pp. 1161–1170. DOI: 10.1109/BigData.2016.7840720.

[9] B. Gao, D. Sheng, L. Zhang *et al.*, 'Star-vp: Improving long-term viewport prediction in 360 videos via space-aligned and time-varying fusion,' in *Proceedings of the 32nd ACM International Conference on Multimedia*, 2024, pp. 5556–5565.

[10] A. Yaqoob and G.-M. Muntean, 'A combined field-of-view prediction-assisted viewport adaptive delivery scheme for 360° videos,' *IEEE Transactions on Broadcasting*, vol. 67, no. 3, pp. 746–760, 2021. DOI: 10.1109/TBC.2021.3105022.

[11] B. Khuong, *The basics of recurrent neural networks (rnns)*, Accessed: 2024-12-02, 2023. [Online]. Available: https://pub.towardsai.net/whirlwind-tour-of-rnns-a11effb7808f.

[12] G. Xiao, M. Wu, Q. Shi, Z. Zhou and X. Chen, 'Deepvr: Deep reinforcement learning for predictive panoramic video streaming,' *IEEE Transactions on Cognitive Communications and Networking*, vol. 5, no. 4, pp. 1167–1177, 2019. DOI: 10.1109/TCCN.2019.2938947.

[13] C. Li, W. Zhang, Y. Liu and Y. Wang, 'Very long term field of view prediction for 360-degree video streaming,' in *2019 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR)*, IEEE, 2019, pp. 297–302.

[14] S. Hochreiter, 'Long short-term memory,' *Neural Computation MIT-Press*, 1997.

[15] N. Yehia, *Understanding long short-term memory networks*, Accessed: 2024-12-02, 2023. [Online]. Available: https://mlarchive.com/deep-learning/understanding-long-short-term-memory-networks/.

[16] L. A. Leiva, Y. Xue, A. Bansal *et al.*, 'Understanding visual saliency in mobile user interfaces,' in *22nd International Conference on Human-Computer Interaction with Mobile Devices and Services*, ser. MobileHCI '20, Oldenburg, Germany: Association for Computing Machinery, 2020, ISBN: 9781450375160. DOI: 10.1145/3379503.3403557. [Online]. Available: https://doi.org/10.1145/3379503.3403557.

[17] L. Itti, 'Automatic foveation for video compression using a neurobiological model of visual attention,' *IEEE Transactions on Image Processing*, vol. 13, no. 10, pp. 1304–1318, 2004. DOI: 10.1109/TIP.2004.834657.

[18] Z. Zheng, Z. Yang, Y. Zhan, Y. Li and W. Yu, 'Perceptual model optimized efficient foveated rendering,' Nov. 2018, pp. 1–2. DOI: 10.1145/3281505.3281588.

[19] L. Itti, C. Koch and E. Niebur, 'A model of saliency-based visual attention for rapid scene analysis,' *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 11, pp. 1254–1259, 1998. DOI: 10.1109/34.730558.

[20] X. Cui, Q. Liu and D. Metaxas, 'Temporal spectral residual: Fast motion saliency detection,' in *Proceedings of the 17th ACM International Conference on Multimedia*, ser. MM '09, Beijing, China: Association for Computing Machinery, 2009, pp. 617–620, ISBN: 9781605586083. DOI: 10.1145/1631272.1631370. [Online]. Available: https://doi.org/10.1145/1631272.1631370.

[21] K. Min and J. J. Corso, 'Tased-net: Temporally-aggregating spatial encoder-decoder network for video saliency detection,' in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, Oct. 2019.

[22] Q. Lai, W. Wang, H. Sun and J. Shen, 'Video saliency prediction using spatiotemporal residual attentive networks,' *IEEE Transactions on Image Processing*, vol. PP, pp. 1–1, Aug. 2019. DOI: 10.1109/TIP.2019.2936112.

[23] A. Boyd, K. W. Bowyer and A. Czajka, 'Human-aided saliency maps improve generalization of deep learning,' in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, Jan. 2022, pp. 2735–2744.

[24] Y. Xu, Y. Dong, J. Wu *et al.*, 'Gaze prediction in dynamic 360 immersive videos,' in *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 5333–5342.

[25] J. Pan, E. Sayrol, X. Giro-i-Nieto, K. McGuinness and N. E. O'Connor, 'Shallow and deep convolutional networks for saliency prediction,' in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2016.

[26] J. Montalvo, Á. García-Martín and J. Bescós, 'Exploiting semantic segmentation to boost reinforcement learning in video game environments,' *Multimedia Tools and Applications*, vol. 82, pp. 10 961–10 979, 2023. DOI: 10.1007/s11042-022-13695-1. [Online]. Available: https://doi.org/10.1007/s11042-022-13695-1.

[27] T. Bubenıcek, 'Using game engine to generate synthetic datasets for machine learning,' 2022.

[28] N. T. Blog, 'Toward a practical perceptual video quality metric,' *Netflix Tech Blog*, 2016. [Online]. Available: https://netflixtechblog.com/toward-a-practical-perceptual-video-quality-metric-653f208b9652.

[29] A. Mahendran, H. Bilen, J. F. Henriques and A. Vedaldi, *Researchdoom and cocodoom: Learning computer vision with games*, 2016. arXiv: 1610.02431 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1610.02431.

[30] I. Sutskever, O. Vinyals and Q. V. Le, 'Sequence to sequence learning with neural networks,' *Advances in neural information processing systems*, vol. 27, 2014.

[31] D. P. Kingma and J. Ba, 'Adam: A method for stochastic optimization,' *arXiv preprint arXiv:1412.6980*, 2014.

[32] H. Yang, Q. Liu and C. Song, 'Adaptive qp algorithm for depth range prediction and encoding output in virtual reality video encoding process,' *PloS one*, vol. 19, no. 9, e0310904, 2024.