VIGOR: Reviving Cloud Gaming Sessions

ZHAOYUAN HE*, The University of Texas at Austin, USA
YIFAN YANG*, Microsoft Research Asia, China
SHUOZHE LI, The University of Texas at Austin, USA
LILI QIU†, Microsoft Research Asia, China
DIYUAN DAI, The University of Texas at Austin, USA
YUQING YANG, Microsoft Research Asia, China

Cloud gaming is a multi-billion dollar industry. A client in cloud gaming sends its movement to the game server on the Internet, which renders and transmits the resulting video back. In order to provide a good gaming experience, a latency below 80*ms* is required. This means that video rendering, encoding, transmission, decoding, and displaying have to finish within that time frame, which is especially challenging to achieve due to server overload, network congestion, and losses. In this paper, we propose VIGOR, a new method to revive cloud gaming sessions by recovering lost or corrupted video frames. Unlike traditional video frame recovery, VIGOR uses game states to enhance recovery accuracy significantly and utilizes partially decoded frames to recover lost portions. We develop a holistic system that consists of (i) efficiently extracting game states, (ii) modifying H.264 video decoder to generate a mask to indicate which portions of video frames need recovery, and (iii) designing a novel neural network to recover either complete or partial video frames. VIGOR is extensively evaluated using iPhone 12 and laptop implementations, and we demonstrate the utility of game states in the game video recovery and the effectiveness of our overall design.

CCS Concepts: • **Information systems** → **Multimedia streaming**; • **Computing methodologies** → **Computer vision**; • **Computer systems organization** → **Real-time system architecture**.

Additional Key Words and Phrases: Cloud gaming, Video frame recovery, Game states, Neural network

ACM Reference Format:
Zhaoyuan He, Yifan Yang, Shuozhe Li, Lili Qiu, Diyuan Dai, and Yuqing Yang. 2024. VIGOR: Reviving Cloud Gaming Sessions. *Proc. ACM Netw.* 2, CoNEXT4, Article 32 (December 2024), 20 pages. https://doi.org/10.1145/3696403

1 INTRODUCTION

Motivation: The cloud gaming market has been experiencing a rapid surge in popularity, with the global cloud gaming market size projected to grow at a CAGR of 64.1% and reach $14.01 billion by 2027 [16]. In cloud gaming, a player's input is transmitted to a remote game server on the Internet, which renders and sends back the resulting video for display on the client side. To provide a satisfactory gaming experience, high bandwidth and low latency are required. For

*Both authors contributed equally to this research.
†Lili Qiu is a corresponding author.

Authors' Contact Information: Zhaoyuan He, zyhe@utexas.edu, The University of Texas at Austin, Austin, Texas, USA; Yifan Yang, Microsoft Research Asia, Shanghai, China, yifanyang@microsoft.com; Shuozhe Li, The University of Texas at Austin, Austin, Texas, USA, shuozhe.li@utexas.edu; Lili Qiu, Microsoft Research Asia, Shanghai, China, liliqiu@microsoft.com; Diyuan Dai, The University of Texas at Austin, Austin, Texas, USA, diyuan.dai@utexas.edu; Yuqing Yang, Microsoft Research Asia, Shanghai, China, yuqing.yang@microsoft.com>.

ACM 2834-5509/2024/12-ART32
https://doi.org/10.1145/3696403

instance, GeForce NOW requires at least 25*Mbps* for 1080p and below 80*ms* latency (40*ms* for the best experience) [13, 26]. However, it is challenging to maintain a high throughput throughout the game session due to fluctuations in Internet performance and server load. If network congestion or server overload occurs, the video frames may not be delivered in time, leading to a disruption in the player's experience.

While TCP retransmission can recover packet losses, its effectiveness is limited in cloud gaming due to the strict real-time requirements. Retransmissions are often too late to be useful, which is why most cloud gaming platforms opt for UDP for faster video transmission [20]. Forward error correction (FEC) can recover losses without adding extra delay, but it is challenging to predict the packet loss rate in advance and properly adjust the amount of FEC to add. Moreover, FEC is expensive: as reported in [45], WebRTC increases the FEC to 50% of original video data under 4% packet loss, and to 80% under 10% packet loss. There are also various proposals to enhance video codecs, but it is challenging to deploy a new codec on a large scale. Therefore, to maximize our impact, we seek an approach that can recover losses without adding significant delay, overhead, or modifying video codecs.

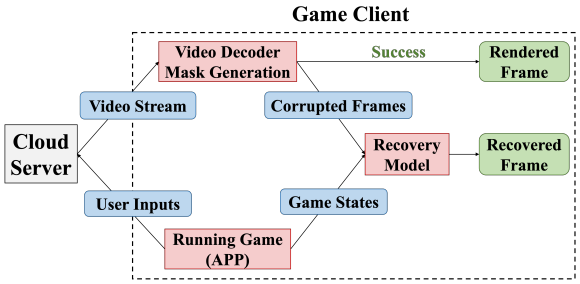


Fig. 1. Overview of VIGOR.

Our approach: Motivated by the observation, we explore how to revive cloud gaming sessions by recovering video frame(s) on the client side whenever the client does not receive them in time. Video prediction and error concealment have been extensively researched in various contexts (*e.g.*, [22, 55, 58, 60, 61, 76, 93]), but in cloud gaming, the client possesses game state information that includes crucial details such as the shapes, sizes, colors, orientations, positions, and movements of game objects, which can be highly beneficial for video recovery purposes.

To leverage game state information for video recovery, there are three critical research questions that need to be addressed: (i) how to efficiently extract and represent the game states, (ii) how to effectively utilize the game states for video recovery, and (iii) to what extent can game states assist in video recovery.

To answer these questions, we develop VIGOR, a novel video recovery system for reviving cloud gaming sessions, shown in Figure 1. First, the game client of VIGOR can efficiently extract game states from a running game app without rendering the whole frame. VIGOR creates a Compute Shader that transforms the 3D vertices of each visible game object into a 2D screen. To speed up game state extraction, VIGOR preloads all objects' vertices into GPU at the beginning of games and creates an array to indicate whether game objects exist in the current frame, thereby minimizing the overhead of uploading information from CPU to GPU. We further experiment with different game state representations, sizes, and resolutions to identify the appropriate configuration that balances the speed and accuracy.

VIGOR then modifies the H.264 video decoder using FFmpeg to generate a mask that indicates which parts of a video frame have decoding errors and require recovery. The game states and corrupted frames are fed into VIGOR's recovery model for video recovery.

Next, VIGOR designs a novel neural network that can leverage game states to recover both complete and partial video frames. VIGOR first computes the optical flow between the current and previous game states and uses it to warp the previous RGB frame to the current one. To improve the image quality, enhancement and inpainting modules are included to not only align pixels but also change colors and generate new content to better restore the image.

In this paper, we propose VIGOR, a hybrid system that splits the rendering between clients and cloud servers. We present a novel design that systematically implements our recovery model on both iPhone 12 and laptop devices, enabling the support of 30 frames per second (FPS) cloud gaming. Through extensive evaluations under diverse network conditions, we demonstrate significant improvements in recovered frames, achieving up to 9.7 dB PSNR. Furthermore, VIGOR effectively leverages game state information, resulting in a remarkable enhancement of up to 5.4 dB PSNR compared to schemes that do not utilize game states. Although we focus on video recovery in this paper, the findings presented in Sec. 8.4 reveal that VIGOR could also reduce bandwidth usage by up to 40%, while maintaining an SSIM greater than 0.9. To the best of our knowledge, VIGOR is the first work to exploit game state information for cloud game video recovery, showcasing a considerable boost in video quality for real-time mobile gaming. This paper does not involve human subjects and has no ethical concerns.

2 RELATED WORK

We classify existing work into the following three areas: (i) cloud gaming systems, (ii) video streaming, and (iii) video recovery.

Cloud gaming systems: [7] provides a comprehensive survey of cloud gaming research. Numerous works focus on optimizing cloud gaming systems, including virtual machine placement [28, 31], distributed game engines [5], GPU sharing [56, 91], cloud scheduling [75], and resource allocation [21, 48, 50]. Some studies [3, 23, 79, 80] use FEC to protect frames from loss, but FEC incurs high overhead [45], causing missed deadlines in fluctuating networks. Outatime [47] renders speculative frames one RTT ahead, while Furion [44] uses cooperative rendering with local foreground rendering and server prefetching. Game engine data has also been used to enhance video encoding motion estimation [62]. Chen *et al.* [9] use collaborative rendering and 3D image warping for dynamic quality adaptation. ZGaming [78] uses an LSTM model for foreground prediction, with depth maps for background prediction, which is vulnerable to packet loss, whereas VIGOR's game states remain effective as they are retrieved from clients.

Video streaming: Existing video streaming systems handle network fluctuations using dynamic rate adaptation, including QDASH [52], FESTIVE [37], buffer-based adaptation [33, 68], and QAVA [11]. [35] and [87] adapt rates based on buffer occupancy, while [34] uses deep reinforcement learning. Several works [8, 36, 49, 65] improve throughput prediction by utilizing session correlations. Extensive research covers LTE video delivery [46, 66]. Regarding 5G, [53] compares adaptive bit rate algorithms, reporting 3.7% to 259.5% higher stall times than 4G/LTE. [29] designs a multicast system for 4K videos over mmWave networks to handle throughput fluctuations. Neural-enhanced streaming methods like NAS [85], LiveNAS [40], NEMO [84], and NeuroScaler [86] use super-resolution to improve transmission efficiency. In contrast, VIGOR focuses on video recovery for cloud gaming, which requires generating new content under strict real-time demands rather than merely enhancing frames.

Video recovery: We classify video recovery into two areas: video prediction and video error concealment. Video prediction uses deep learning models like convolutional networks [2, 69, 74, 76], recurrent networks [67, 93], generative adversarial networks (GANs) [42], and diffusion models [27, 32, 81, 83]. While GANs produce realistic images, they struggle with accurate frame prediction.

Diffusion models are effective but too computationally expensive for real-time use on mobile devices due to their complexity. For video error concealment, neural networks are used to recover corrupted frames from previous ones. [60] predicts optical flow to reconstruct degraded frame areas, and [61] employs capsule networks for motion-compensated error concealment. GRACE [12] trains a neural video codec for packet loss recovery, but inaccuracies in the motion vector estimates from the neural encoder can cause significant errors in the recovered frames. In contrast, VIGOR extracts precise game states to compute optical flow, improving recovery reliability. NERVE [30] uses binary point code extraction, but its extra transmission and lack of adaptability introduce latency, limiting recovery performance.

Summary: VIGOR differs from the existing work as follows: (i) VIGOR is the first one that extracts and leverages the game state for complete or partial video frame recovery and demonstrates significant benefits of game states, and (ii) unlike the existing work, which focuses only on the video reconstruction accuracy, VIGOR accurately recovers videos in real-time on mobile devices.

3 MOTIVATION

(a) Viking Village (b) Nature (c) Corridor (d) Viking Village+

Fig. 2. High-quality Unity games.

3.1 Local Rendering

Network performance can vary significantly [46, 53, 66], making local rendering on the client side practical when conditions degrade. To assess this, we measure resource usage and frame rate on two devices: (i) iPhone 12 (Apple A14 Bionic chipset, hexa-core CPU, 4-core GPU) and (ii) a laptop (Intel Core i5-9300H, GeForce GTX 1650 GPU). According to [15], iPhone 12 and newer models accounted for over 95% of U.S. iPhone purchases in Q1 2024. Additionally, Android smartphones like Samsung Galaxy S24 and Xiaomi 14 held over 88% market share in Q1 [18], with comparable or better CPU/GPU performance than the iPhone 12 [25]. Thus, most users' smartphones likely meet or exceed iPhone 12's performance. We test three games—Viking Village [73], Nature [54], and Corridor [17]—high-quality Unity apps designed for high-end PCs (Figure 2). Table 1 shows that these games achieve only 17-23 FPS on the iPhone 12, below the typical 30 FPS requirement. The laptop, with a more powerful CPU and GPU, achieves over 30 FPS, but struggles as game complexity increases. To validate this, we create an expanded version of Viking Village (Viking Village+), adding more objects (Figure 2(d)), where the laptop achieves only 18 FPS. These results suggest that local rendering is often impractical, motivating us to explore alternatives for recovering corrupted frames caused by network or server delays.

3.2 Limitations of FEC

FEC is widely used for video frame recovery. To evaluate its performance, we set up a live WebRTC streaming session with FlexFEC and vary packet loss rates (1%, 3%, 5%, and 10%) under a WiFi network, with WebRTC's Google Congestion Control (GCC) and retransmissions enabled. Figure 3 shows that increasing FEC redundancy initially reduces pixel loss by improving packet recovery, but excessive redundancy burdens the network, leading to higher pixel loss. Packets are considered lost if they miss the playout deadline (*e.g.*, 80ms). Optimal FEC redundancies for 1%, 3%, 5%, and 10% loss rates are 20%, 25%, 30%, and 40%, respectively, reflecting the high cost of FEC as noted in prior works [45, 51, 59]. The results also show that FEC alone cannot achieve zero frame loss, even at

optimal levels. Some works [45, 51, 59] reduce FEC overhead but face challenges with unpredictable networks. Therefore, more efficient, adaptive recovery methods are needed.

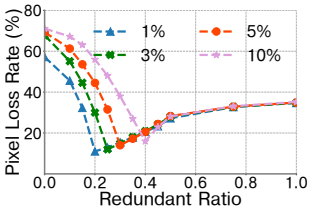


Fig. 3. Average pixel loss rate under different packet loss rates and FEC redundancy ratios.

Table 1. Frame rate when running different games

Machine	Game Name	FPS
iPhone 12	Viking Village	20
	Nature	23
	Corridor	17
Laptop	Viking Village+	18

3.3 Partial Recovery

To analyze pixel loss distribution, we plot the CDF of average pixel loss rates from the experiment in Sec. 3.2. Using the H.264 codec in FFmpeg, the decoder processes remaining macroblocks when some are missing, leading to partial frame loss, requiring error concealment. We enable H.264's default method, which averages pixel values for missing intra-macroblocks and uses boundary-matching-based motion vector recovery for missing inter-macroblocks. Averaging causes blurriness, and motion vector recovery struggles with scene changes [92]. Figure 4 shows that at 1% packet loss, 25% of frames are completely lost, and 50% are partially corrupted, increasing with higher loss rates. When reference frames are lost, all dependent frames are also lost. This emphasizes the need for effective video recovery methods for both partial and complete frame loss.

3.4 Video Prediction

Video prediction can help recover lost frames, but many state-of-the-art models are too heavy for real-time inference on the laptop (*e.g.*, within 33*ms*). For example, CrevNet [89] and PreRNN+ [77] take 600*ms* and 3*s*, respectively, to predict 1080p frames. An alternative is to calculate the optical flow between the last two frames and warp the previous frame. However, even state-of-the-art optical flow models are not fast enough for real-time 1080p videos; for instance, SPyNet [57] takes 250*ms* to compute optical flow between two frames on the laptop. Accurately predicting video frames is challenging due to uncertainties in future events. For example, CrevNet [89] achieves only 0.49 SSIM and 19.2dB PSNR for game video prediction, which is insufficient for a satisfactory gaming experience. Meanwhile, we observe that game video frames are rendered based on game states. This allows us to leverage the game states to significantly improve prediction accuracy.

3.5 Summary

In this section, we highlight the challenges of real-time local rendering and video recovery, noting that current game video prediction accuracy is insufficient. We also show that FEC incurs high bandwidth costs and cannot achieve zero frame loss. Additionally, partial frame corruption is common with packet loss. These findings motivate the development of an efficient video recovery algorithm to address both partially corrupted and completely lost frames.

4 EXTRACT AND REPRESENT GAME STATES

In this section, we present how VIGOR extracts and represents the game states. We use the games developed with Unity [70], which is the most widely used game engine, with 38% of developers using it as their primary engine [64]. Our high-level approach is compatible with games from any game engine, though our script is more easily applied to Unity-based games. Below we first provide a brief overview of Unity and then describe our game state extraction.

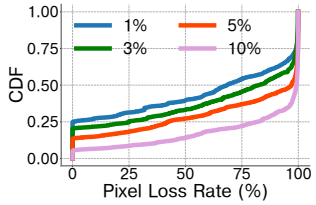


Fig. 4. Average pixel loss rate of each frame under different network loss rates.

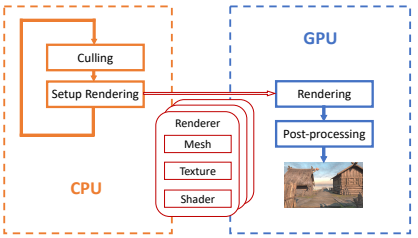


Fig. 5. Simplified Rendering Process in Unity

4.1 Background

Unity [70] is a cross-platform game engine developed by Unity Technologies. It is primarily used to develop video games and simulations. At a high level, the Unity rendering pipeline consists of culling, rendering, and post-processing [71]. Figure 5 shows a simplified rendering process in Unity.

Culling: For each frame, a camera uses frustum culling and occlusion culling to exclude objects outside its view, improving efficiency by avoiding rendering unnecessary objects. Subsequently, the CPU uploads visible renderers and camera matrices to the GPU for rendering.

Rendering: The rendering step involves creating geometry, adding lighting effects, and projecting the 3D environment onto the 2D screen based on the camera configuration. A renderer consists of meshes (defining object geometry with vertices and triangles), textures (specifying object color and transparency), and shaders (programs calculating pixel colors based on lighting and material).

Post-processing: Unity offers post-processing effects, such as blurring the background and moving objects, balancing colors around white areas, adding fog effects, etc. They are used to create specialized visual effects.

4.2 Game States

VIGOR extracts game states from a running game app, including object locations, speeds, rotations, and sizes, using these along with previous frames to enhance video prediction.

Algorithm 1 shows the pseudo-code for VIGOR's game states extraction. On the CPU side, VIGOR involves extracting renderers, performing frustum culling to filter visible objects, and then sending relevant data to the GPU—this includes the camera's Projection (P), View (V), and object's Transform (T) matrices, along with all visible vertices. T transforms from the object's local space to the global space, V converts the global space to the camera view, and P specifies how the camera maps a 3D world onto a 2D image. Different from the CPU processing in Figure 5, this process captures essential game states for each frame without needing to render the entire frame, thereby avoiding much overhead on the client.

Algorithm 1 Game States Extraction

```
CPU:
for frame f = 1, 2, ... do
    for object i = 1, 2, ..., N do
        Check the bounding box against the frustum of the camera to skip invisible objects
    end for
    Send P, V, T matrix and all visible vertices to the Compute Shader running on GPU
end for

GPU:
for all vertices in parallel do
    Perform MVP matrix multiplication
    Transform each 3D vertex from the local space to the 2D screen space
    Filter out the points outside the 2D screen
end for
Send the game states back to CPU
```

Compute shaders are programs running on the GPU outside the rendering pipeline. They can be used for massively parallel GPU computation. VIGOR develops its own compute shaders to derive game states by transforming 3D vertices of visible objects onto a 2D screen, due to Unity's lack of APIs for accessing intermediate results. This involves calculating the Model View Projection

(MVP) matrix from the camera's Projection (P), View (V), and object's Transform (T) matrix using $\mathbf{MVP}_{f,i} = \mathbf{P}_f \times \mathbf{V}_f \times \mathbf{T}_{f,i}$ where f is the frame index and i is the game object index. MVP converts each object from the local space to the clip space. Then VIGOR specifies the 2D screen size to perform a viewport transform and generates the final game state frame on the 2D screen. Figure 6(a) and 6(b) are a RGB frame and a game state frame at a resolution of 270×480. In the following context, the terms game state frame and game states are used interchangeably.

As mentioned above, game state extraction involves the following steps: (i) perform culling, (ii) upload visible vertices of game objects from CPU to GPU, (iii) perform the matrix multiplication in parallel, and (iv) download the game states from GPU to CPU. The following two equations show the time for generating 270×480 game states in Viking Village on the iPhone 12 and laptop. This is even slower than performing local rendering, which is not acceptable.

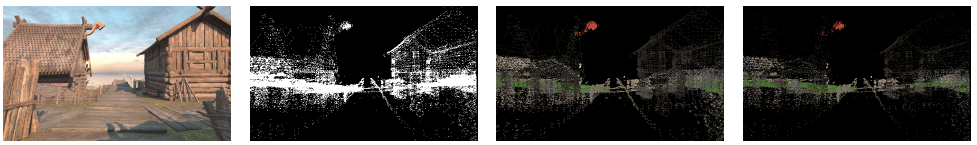
$$\underset{(435.8ms)}{T_{gs\_iphone12}} = \underset{(1.5ms)}{T_{culling}} + \underset{(420ms)}{T_{CPU \rightarrow GPU}} + \underset{(12.8ms)}{T_{multiplication}} + \underset{(1.5ms)}{T_{GPU \rightarrow CPU}}$$

$$\underset{(237.5ms)}{T_{gs\_laptop}} = \underset{(3.3ms)}{T_{culling}} + \underset{(230ms)}{T_{CPU \rightarrow GPU}} + \underset{(4ms)}{T_{multiplication}} + \underset{(0.2ms)}{T_{GPU \rightarrow CPU}}$$

4.3 Speed-up Game State Extraction

We identify the transfer of millions of 3D vertices from CPU to GPU as a major latency bottleneck. To optimize this, VIGOR preloads vertices into the GPU at game start, avoiding per-frame uploads. Instead, VIGOR updates a status array to indicate the presence of game objects and pass it, along with the P, V, and T matrices, to the Compute Shader for matrix multiplication. This reduces $T_{CPU \rightarrow GPU}$ to $0.2ms$ on the iPhone 12 and $0.08ms$ on the laptop. The total size of all game objects' vertices is around 200MB in Viking Village and 630MB in Viking Village+, which can fit into GPU's VRAM on the iPhone 12 and the laptop.

4.4 Game State Representation

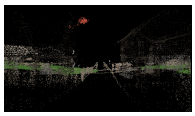


(a) Original Frame (b) No Color (c) Colored (d) Downsample 5x

Fig. 6. An example of game states.

Colored game states: Colored game states give richer information and are likely to help video frame prediction. As an example, Figure 6(b) and 6(c) show a gray-scale game state and a colored one. We can more easily see different objects from the colored game state. However, colored states increase I/O operations due to the need for 3 channels (RGB), each with 4-byte pixel values, leading to increased $T_{GPU \rightarrow CPU}$ (from $1.5ms$ to $4.2ms$ on the iPhone 12). To avoid the delay increase, VIGOR uses a HashMap to store original colors of game objects, allowing the GPU to send back colored indices instead of actual pixels. The CPU then matches these indices to colors in the HashMap, recreating RGB game states without additional latency. This method only slightly increases memory usage (e.g., 4.4KB for Viking Village's 1,500 objects). VIGOR also utilizes the depth information of 3D vertices to filter out the farther vertex if two vertices from different game objects are mapped to the same point on the 2D screen.

Game state resolution: To reduce the matrix multiplication time on GPU, VIGOR downsamples the vertices in the game states. We try different downsampling ratios and measure the computation time. Table 2 shows that downsampling by a factor of 2, 5, and 10 reduces $T_{multiplication}$ to $7.9ms$, $4.3ms$, and $3.6ms$ on the iPhone 12, and reduces to $2.5ms$, $1.3ms$, and $1.1ms$ on the laptop, respectively.



(a) 270 × 480

(b) 64 × 128

(c) 64 × 64

Fig. 7. Game states in different resolutions.

Table 2. $T_{multiplication}$ with different downsampling ratios and $T_{GPU \to CPU}$ with different sizes

	$T_{multiplication}$			$T_{GPU \to CPU}$	
Downsampling	iPhone 12	Laptop	Size	iPhone 12	Laptop
1×	12.8ms	4.0ms	270 × 480	1.5ms	0.2ms
2×	7.9ms	2.5ms	64 × 128	0.8ms	0.1ms
5×	4.3ms	1.3ms	64 × 64	0.6ms	0.08ms
10×	3.6ms	1.1ms			

Figure 6(d) shows an example of game states with 5× downsampling ratio. In order to reduce $T_{GPU \to CPU}$, VIGOR also varies the resolution of the game states. Table 2 shows that reducing the frame size from 270 × 480 to 64 × 128 and 64 × 64 reduces $T_{GPU \to CPU}$ to 0.8ms and 0.6ms on the iPhone 12, and reduces to 0.1ms and 0.08ms on the laptop. Examples in Figure 7 show that despite different resolutions, objects in game states remain clearly visible. For visualization purposes, these frames are displayed at the same size, though actual resolution is used for inference.

4.5 Summary

VIGOR's optimization reduces the latency of game state extraction to the following two equations.

$$\underset{(5.9-16ms)}{T_{gs\_iphone12}} = \underset{(1.5ms)}{T_{culling}} + \underset{(0.2ms)}{T_{CPU \to GPU}} + \underset{(3.6-12.8ms)}{T_{multiplication}} + \underset{(0.6-1.5ms)}{T_{GPU \to CPU}}$$

$$\underset{(4.5-7.6ms)}{T_{gs\_laptop}} = \underset{(3.3ms)}{T_{culling}} + \underset{(0.08ms)}{T_{CPU \to GPU}} + \underset{(1.1-4ms)}{T_{multiplication}} + \underset{(0.08-0.2ms)}{T_{GPU \to CPU}}$$

5 MASK GENERATION

When network performance deteriorates, video streams may be partially lost, leading to corrupted frames on the client side. Identifying the corrupted frame parts is crucial for targeted recovery.

Video frames, including key frames (I-frames) and change-based frames (P-frames and B-frames), can have decoding errors due to corruption in either the frame's packets or its reference frame. In order to properly generate a mask that indicates which parts of the frame are corrupted, we need to understand the video codec. We focus on the H.264 codec, used by platforms like Stadia and GeForce Now. FFmpeg, an open-source video codec, is used for our implementation.

In H.264, a coded picture comprises macroblocks of three types: I macroblocks (intra-predicted), P macroblocks (inter-predicted from a reference frame), and B macroblocks (similar to P but predicted from one or two reference frames). All macroblocks range from 4 × 4 to 16 × 16. The decoder stops decoding subsequent macroblocks if earlier ones are corrupted, thereby giving us an accurate position indicating which macroblocks are corrupted. VIGOR generates a mask for each sub-macroblock (minimum size 4 × 4), considering both the macroblock and its reference block, to indicate decoding success. An example in Figure 8 shows a corrupted frame and its mask, with white indicating correct decoding and black indicating corruption.

This mask enables the use of partially decoded frames for video recovery, adding minimal overhead as it utilizes intermediate decoding variables without extra computation. The mask is a gray-scale array that takes around 2MB of memory. To accommodate frames' reference dependencies, VIGOR caches multiple masks (*e.g.*, 10) for future use.

(a) Corrupted Frame (b) Mask

Fig. 8. An example of a corrupted frame and the corresponding mask.

6 VIDEO RECOVERY MODEL

There have been lots of works (*e.g.*, [39, 55, 90]) using previous frames to recover a video frame. In cloud gaming, we can further leverage current game states along with previous video frames to improve the performance since the game states contain crucial information regarding current objects' shapes, sizes, colors, orientations, positions, movements, etc.

Game state changes strongly correlate with RGB frames (from Unity's rendering logic), so they help our network to learn the mapping to RGB space optical flow. Our intuition is that the pixel-wise shifting between the two game states at different time steps is similar to that between the corresponding two RGB frames. Therefore, we can first compute the optical flow between the current and previous game states, and use it to warp the previous frame to predict the current one.

6.1 Estimating Optical Flow

The optical flow captures the movement of the brightness pattern and approximates the motion field. Let $u(x, y)$ and $v(x, y)$ denote the 2D optical flow for a given point (x, y) in a frame. Under the assumption of the "intensity constancy", we have $I(x + \Delta x, y + \Delta y, t + \Delta t) \approx I(x, y, t)$. Through a Taylor series expansion, $I(x + \Delta x, y + \Delta y, t + \Delta t) = I(x, y, t) + \frac{\delta I}{\delta x}\Delta x + \frac{\delta I}{\delta y}\Delta y + \frac{\delta I}{\delta t}\Delta t$. Hence we have $\Delta I v + I_t = 0$ where $v = (\Delta x, \Delta y)$ is the optical flow, $\Delta I = (I_x, I_y)$ is the spatial gradient, and I_t is the temporal gradient. By deriving ΔI and I_t from a pair of video frames, we get the optical flow v.

Optical flow can be estimated by phase correlation, block-based, differential methods, discrete optimization, and deep learning approaches. We choose SPyNet [57], one of the most widely used optical flow networks due to its high efficiency and accuracy.

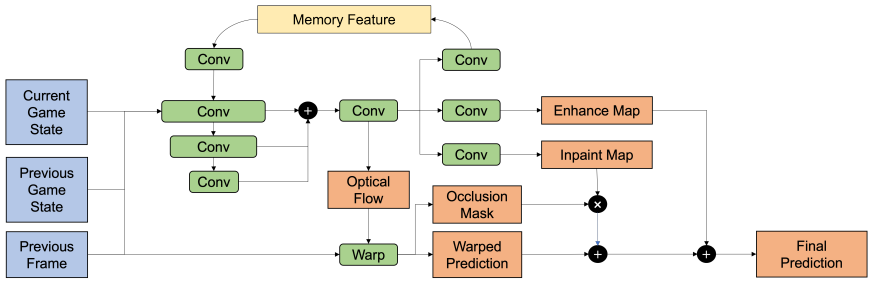


Fig. 9. Model architecture diagram.

Figure 9 shows VIGOR's model architecture. It initializes the pyramidal convolution backbone with a SPyNet pre-trained on Sintel [6]. It uses the current and previous game states to capture optical flow. It then downsamples the input 5 times through a pyramid structure and accumulates the predicted optical flow layer by layer to get the final optical flow.

6.2 Refining & Warping Using Optical Flow

The optical flow between the two consecutive game states can be used to warp the previous RGB frame to the current one. However, the optical flow between the two game states differs from the

optical flow between the two corresponding RGB frames in the following way: (i) game states have a finite number of scatter points and their frames are more sparse, (ii) the movement of the scatter points are not the same as the movement of brightness patterns in RGB frames, and (iii) the resolution of game states is usually much lower than RGB frames for the sake of computational efficiency, so the resulting flow cannot be used directly for warping.

For the output optical flow, VIGOR uses bilinear interpolation and grid sample-based warping to create an aligned prediction from the previous RGB frame. This fully differentiable process enables gradient propagation to optimize the optical flow network and produce accurate results.

6.3 Enhancement and Inpainting

Warp-based video frame prediction can only predict the visible content in the previous RGB frames, and cannot modify the color gap for the new frame or generate content that does not exist in previous frames. Therefore, VIGOR improves the optical flow network by adding the following inputs to enable the generation of new content: (i) previous frame to add unavailable details in the game states, (ii) current partially decoded frame as a reference to align the generated content to the received partial frame, and (iii) memory feature, including historical prediction information, fed into the current frame prediction to ensure temporally smooth generated frames.

After the pyramid network, VIGOR applies two sets of convolutions to generate the Enhance Map and Inpaint Map, as shown in Figure 9. These maps are scaled to the size of RGB frames using Pixel Shuffle [63]. The Enhance Map bridges the gap between the warped frame and the ground truth, enhancing meaningful content and ignoring empty areas. The Inpaint Map generates new content when the warping process lacks references, focusing on inpainting missing parts. It is multiplied by the occlusion mask, which is 1 where the warped prediction is non-empty and 0 otherwise. The final prediction is obtained by adding this result to the Enhance Map and overwritten by incorporating the partially decoded content of the input frame.

We quantify the difference between the predicted and actual RGB frames using the following loss function: $\mathcal{L}^t = \mathcal{L}^t_{Pix} + \alpha \mathcal{L}^t_{distill}$. α is set to 0.1. $\mathcal{L}^t_{Pix} = C(\widehat{\mathbf{I}}^t, \mathbf{I}^t)$ and $C(\widehat{\mathbf{I}}^t, \mathbf{I}^t) = \sum_{\mathbf{i}}^{\mathbf{W}} \sum_{\mathbf{j}}^{\mathbf{H}} \sqrt{(\widehat{\mathbf{I}}^t_{ij} - \mathbf{I}^t_{ij})^2 + \epsilon^2}$ where $\widehat{\mathbf{I}}^t$ and $\mathbf{I}^t$ denote the predicted and actual frames at time t, respectively. $\mathbf{W}$ and $\mathbf{H}$ denote the width and length of the input video, and ϵ is a small constant set to $1e^{-12}$. We use Charbonnier loss [43] as the loss function C. The Charbonnier loss is a differentiable variant of $\mathcal{L}_1$ distance, which is shown to perform well for generation tasks [4, 19, 24]. We also add a distillation loss[41, 63] $\mathcal{L}^t_{distill} = C(\mathbf{O}^t_S, \mathbf{O}^t_{fix})$ to the output optical flow $\mathbf{O}^t_S$. $\mathbf{O}^t_{fix}$ comes from the output of the fixed SPynet we used for initialization in order to maintain the warping stability. We find that without this item, $\mathbf{O}^t_S$ tends to favor making the warped prediction be the final prediction to reduce loss, which renders the inpainting module ineffective.

7 SYSTEM IMPLEMENTATION

Figure 1 illustrates VIGOR's system architecture, which includes a game client that generates game states efficiently without rendering entire frames. A video decoder creates a mask to identify areas in video frames requiring recovery. The game states and corrupted frames are fed to VIGOR's recovery model for video frame recovery. We also devise strategies for when to run the recovery model and how to optimize video quality.

Easy to deploy: VIGOR's game state extraction is easy to deploy with two scripts (around 400 lines of code). The first script preloads vertices into the GPU and transmits matrices to extract game states per frame, while the second uses a Compute Shader on the GPU to generate game states. These scripts work for any new Unity game without modification. Developers can attach them as modules to the main camera via Unity's UI, without needing to understand the source

code. The client game app can also be transformed into a Chrome browser extension [14] for easy deployment, similar to platforms like Stadia and GeForce NOW. Additionally, VIGOR's modifies the video codec to generate masks, which can be compiled into FFmpeg during video decoding.

Speeding up inference time: The inference time of VIGOR's recovery model depends on the game state size, adapted to the iPhone 12 and laptop capabilities. On the iPhone 12, VIGOR uses a 64×128 game state resolution, downsampled 5 times, while on the laptop, VIGOR uses a 270×480 resolution without downsampling. The model supports various input sizes by upsampling optical flow to match the target output size, ensuring efficient use of hardware.

On the iPhone 12, VIGOR uses CoreML for optimized performance, converting our pretrained Pytorch model to CoreML, which performs faster than ONNX, Pytorch Mobile, and TensorFlow Lite. Due to limited GPU support for grid sampling, VIGOR uses Metal Performance Shaders (MPS) for a custom, GPU-accelerated grid sample layer. Additionally, VIGOR performs warping at a reduced 270p scale, thereby reducing the warping time from $29ms$ to $5ms$. We also use FP16 precision to further decrease inference time to $22ms$ on the iPhone 12. On the laptop, VIGOR uses Pytorch with CUDA on Linux, optimizing with TVM [10], which reduces inference time by 50%, achieving a final time of $25ms$.

Foreground interactions and deformable objects: Furion [44] suggests that a game frame consists of a predictable, detail-rich background and an unpredictable, lightweight foreground. The background is better rendered on the server, while the client renders foreground interactions. VIGOR follows this approach by rendering foregrounds locally and recovering lost or corrupted backgrounds, then combining them for the final frame.

VIGOR preloads rigid objects onto the GPU and uses our model to recover frames, as their vertices remain unchanged. For deformable objects requiring frequent vertex updates, VIGOR uses Furion's cooperative rendering strategy to render them locally, which takes around $2ms$ on the iPhone 12 and $1.5ms$ on the laptop, suitable for 30 FPS, shown in Figure 10. This allows us to efficiently handle both rigid and deformable objects.

Scheduling recovery: Instead of waiting for a video frame to be completely lost before initiating recovery, which causes significant delays, VIGOR overlaps client-side game state extraction with server-side processing and network transmission. This approach, however, requires the client to extract game states without confirmation of the current video frame's timely arrival. To efficiently manage client resources and minimize end-to-end delay, VIGOR initiates game state extraction only if the preceding video frame was lost since network losses are bursty according to both previous measurement studies [82, 88] and our own measurement. Our analysis of network traces, as shown in Table 3, indicates a 95% probability of current frame loss if the previous frame was lost.

End-to-end latency: VIGOR's strategy enables neural recovery without waiting for video packets received, with recovery latency being the sum of game state extraction time and model inference. On the iPhone 12, using 64×128 game states downsampled by 5 times, VIGOR achieves $7ms$ for game state extraction and a total latency of $29ms$. For the laptop, using 270×480 resolution without downsampling, game state extraction takes $5ms$, totaling $30ms$ in latency.

End-to-end latency in cloud gaming, from user input to frame display, has an acceptable range of 80-150ms [9, 13, 26, 80], depending on the game type. We set $80ms$ as our maximum latency budget. To maintain latency within this budget, we implement a timeout mechanism for video recovery. If a partially corrupted frame is received and processed within this timeout, VIGOR performs video concealment for recovery. If not, video prediction is executed to avoid exceeding the deadline.

8 PERFORMANCE EVALUATION

In this section, we present our evaluation methodology and performance results.

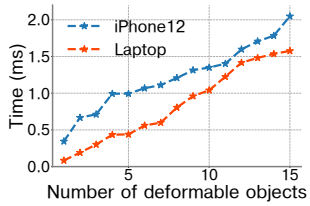


Fig. 10. Local rendering time for deformable objects.

Table 3. Network traces

	4G	5G	WiFi	LEO
Amount of traces	145	152	150	146
Avg. Duration (s)	320	390	300	300
Avg. Throughput (Mbps)	32.5	61.7	72.8	28.9
Avg. Loss rate (%)	3.2	2.3	0.9	9.4

8.1 Evaluation Methodology

Datasets and metrics: We use Viking Village [73], Nature [54], Corridor [17] on the iPhone 12, and Viking Village+ on the laptop. Each game video lasts around 5 mins and they are collected by different players. There are 50,000 training frames and 10,000 testing frames for each game. We quantify the quality of our recovered video frames using two widely used video quality metrics: SSIM and PSNR. Higher SSIM and PSNR values indicate better video quality.

Network traces: Table 3 shows the network traces we use. We measured the downlink throughput using *iperf* from an Azure server located in the central U.S. to a local client over the Internet, where we varied the wireless hop to use WiFi, 4G, and 5G networks. *iperf* generates TCP streams and provides the number of retransmissions and throughput. We compute the packet loss rate by dividing the number of retransmissions by the total number of transmissions. The 4G and 5G traces include static and walking scenarios. We also move the client randomly to add mobility to the WiFi traces. The low earth orbit (LEO) satellite network traces were collected from the StarLink network in January 2023 using a StarLink RV ground station in the west coast of U.S. under a static case. To measure the packet loss rate, we ping the server every 10ms and collect the average ping drop rate.

Evaluation strategies: We use collected network traces to delay packets based on their recorded timestamps before sending them to the client. The client decodes the video and recovers corrupted frames as needed. We evaluate video recovery in two scenarios: one assumes all previous frames are received correctly, recovering each frame individually; the other recovers frames using previously rendered or recovered frames, feeding the recovered frame to the model for subsequent recovery. These strategies complement each other: the former focuses on individual frame recovery, while the latter addresses realistic network losses. Using network traces for evaluation ensures consistent loss conditions for all recovery algorithms. Packet loss is defined as missing the deadline, which is set by adding the maximum tolerable latency (80ms) to the inter-frame spacing (33ms for 30 FPS).

Implementation details: We train for 100,000 iterations on an 8-GPU Tesla V100 machine with a batch size of 16. The learning rate is set to $1e^{-5}$ and adjusted using cosine decay.

8.2 Performance Results

Impact of game states: We compare the following schemes for different games to analyze the impact of game states: (i) VIGOR (with game states), (ii) without game states, (iii) reusing the last frame as the recovered frame, and (iv) ECFVI [38], a flow-guided video inpainting model similar to our optical flow-based recovery. NERVE [30] highlights ECFVI's strong performance, particularly for long sequences of recovered frames, making it a solid baseline for neural video recovery. Note that here we assume the previous frames are received correctly. In (ii), the last two frames are used instead of game states as input for optical flow estimation. In (iv), we modify ECFVI, originally limited to 240p, by adding convolutional and pixel shuffle layers to upscale to 1080p, training it on our dataset and evaluating it at the same epoch as our model.

Figure 11 illustrates the video prediction results. For Viking Village, (i) yields improvements in SSIM of 0.06, 0.1, and 0.012, and in PSNR of 3.5dB, 4dB, and 0.7dB, compared to methods (ii), (iii),

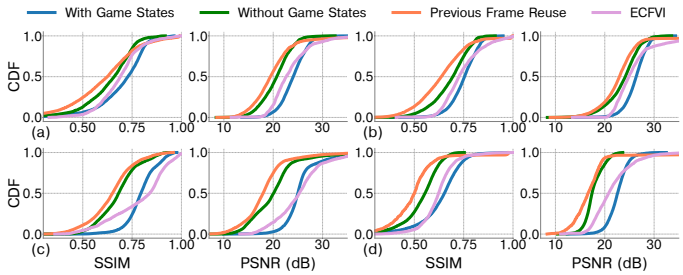


Fig. 11. Video Prediction results for different games. (a) Viking Village. (b) Nature. (c) Corridor. (d) Viking Village+

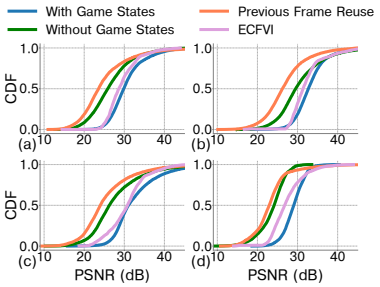


Fig. 12. Partial recovery results for different games. (a) Viking Village. (b) Nature. (c) Corridor. (d) Viking Village+

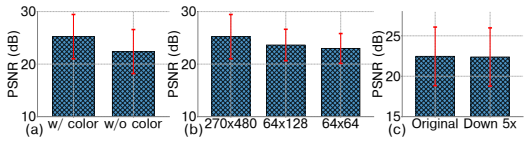


Fig. 13. Performance results for game states. (a) w/ and w/o color. (b) different sizes. (c) different downsampling ratios.

and (iv), respectively. For Nature, we see SSIM boosts of 0.07, 0.12, and 0.008, and PSNR increases of 2.6dB, 3.1dB, and 0.4dB. Corridor benefits with SSIM increases of 0.11, 0.15, and 0.006, and PSNR gains of 5.6dB, 7.3dB, and 0.3dB. Lastly, Viking Village+ shows SSIM enhancements of 0.09, 0.14, and 0.016, and PSNR improvements of 4.9dB, 5.6dB, and 0.9dB.

We make the following observations. First, using game states significantly improves video quality over recovery without game states. Second, reusing previous frames only performs slightly worse than recovery without game states due to little change in consecutive video frames. Third, VIGOR's light-weight video recovery model based on game states yields even better results than ECFVI, a much bigger model. We implement ECFVI with the same CoreML optimization and test it on a MacBook Air because its high memory requirements prevent deployment on an iPhone 12. On the MacBook Air, ECFVI takes 6s for inference, while VIGOR's model takes only 18ms. This demonstrates VIGOR's efficiency and the benefit of incorporating game states.

We also measure the average PSNR values for upsampling frames from 270p to 1080p across the four games, obtaining 27.6dB, 30.2dB, 28.7dB, and 26.1dB, respectively. These results indicate that VIGOR's video prediction approach only slightly underperforms compared to the upsampling method, despite the quality values being relatively modest.

Figure 12 shows the recovered video quality of partially corrupted frames, with PSNR results for brevity. Partially corrupted frames have higher recovered quality than completely lost frames because they retain some original content. Figure 14 visualizes VIGOR's recovery model using different masks for various games, demonstrating that VIGOR effectively recovers and stitches partially corrupted frames for accurate results. In contrast, predicting optical flow without game states leads to blurred and corrupted recovered frames.

Impact of game state representation: Figure 13 explores the effect of game state representations on video quality, focusing on color, size, and downsampling ratio. We observe the followings: 1) Adding color improves video quality by 2.9dB in PSNR, aiding in object distinction; 2) Among the evaluated game state sizes (270×480, 64×128, and 64×64), larger sizes enhance performance, yet a 64×64 game state still delivers a decent 23dB PSNR; 3) Downsampling game object vertices by 5 times marginally impacts accuracy but significantly reduces computation time by over 66% on both the iPhone 12 and the laptop.

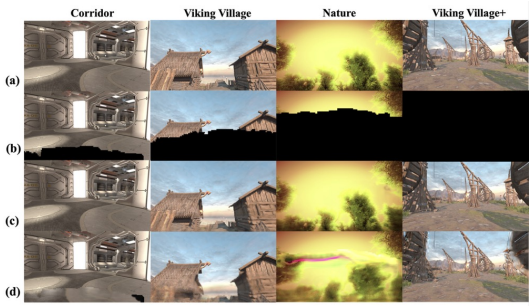


Fig. 14. Visualizations analysis of VIGOR's recovery model using different masks for different games. The first three columns are the results on the iPhone 12, and the last column is the result on the laptop. (a) Ground truth. (b) Partial frame extracted from the decoded frame using mask. (c) Recovered frame. (d) Recovered frame without game states.

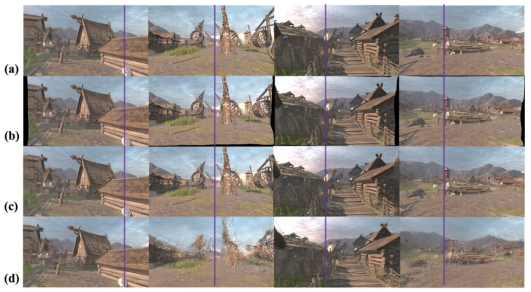


Fig. 15. Visualizations analysis of enhancement and inpainting for Viking Village+ on the laptop. A purple auxiliary line is added for alignment with the ground truth. We make a whole frame prediction. (a) Ground truth. (b) Warped result. (c) Recovered frame. (d) Recovered frame without game states.

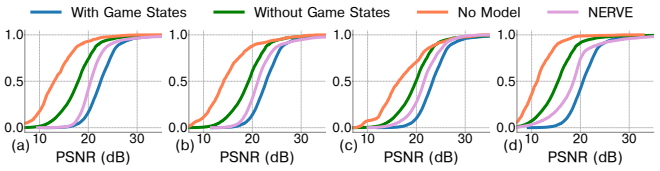


Fig. 16. Performance results for different network traces. (a) 4G. (b) 5G. (c) WiFi. (d) LEO.

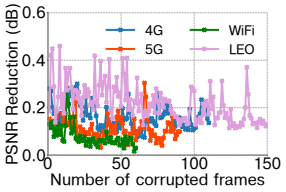


Fig. 17. PSNR reduction with the number of consecutive corrupted frames.

Enhancement and generation: In Figure 15, row (b) demonstrates the limitation of warping, which can only move existing pixels, resulting in black borders in areas lacking references from previous frames. Row (c) illustrates how the inpainting module effectively fills these black edges by using game state information for object position hints. Row (d) shows that inaccurate flow causes misalignment in warping, but our enhancement module corrects the color of the mispositioned object. However, regenerating from scratch leads to blurred and inaccurate predictions. Thus, game states are crucial for generating high-quality optical flow, essential for accurate feature alignment.

Impact of different network conditions: High packet loss (above 5%) can severely disrupt Internet connectivity, affecting page loads, media playback, and online gaming [1]. We evaluate network traces (4G, 5G, WiFi, LEO) under packet loss rates from 0.9% to 9.4% and compare four recovery schemes: (i) VIGOR (with game states), (ii) without game states, (iii) without recovery model (reusing previous frames), and (iv) NERVE [30] (using binary point codes). Figure 16 shows that scheme (i) outperforms others, with improvements of up to 5.4dB over (ii), 9.7dB over (iii), and 2.2dB over (iv). The gain of (i) over (ii) is slightly better compared to Figure 11(a) since (ii) uses the last two frames for recovery, leading to more error accumulation. Scheme (iii) performs worst, degrading rapidly with bursty losses. NERVE outperforms (ii) and (iii) due to the usage of binary point codes but is less effective than VIGOR, as game states provide more accurate optical flow for recovery, especially under bursty losses. Additionally, NERVE has a higher latency overhead from extracting and transmitting binary point codes, leading to more packet loss in cloud gaming scenarios. Overall, PSNR decreases with increasing loss rates, but using game states reduces error accumulation, resulting in better recovery.

Impact of scheduling recovery: If a frame is lost or misses its deadline, the client initiates game state extraction using the last rendered frame for recovery. If frame A is missed, its successor is recovered using A's predecessor. Game state extraction for both A's successor and predecessor

Table 4. Comparison with forward error correction (FEC)

Network Traces	4G		5G		WiFi		LEO	
Recovery Type	Our	40% FEC	Our	35% FEC	Our	25% FEC	Our	70% FEC
Avg. pixel loss rate (%)	71	28	63	22	57	12	83	41
Avg. PSNR (dB)	31.65	29.01	33.53	31.58	37.51	36.8	27.95	23.26

happens simultaneously, leveraging cached user inputs. On the iPhone 12, this parallel process adds only $0.8ms$, a negligible increase. Figure 17 shows VIGOR's minimal PSNR reduction during long frame loss bursts (0.17dB for 4G, 0.13dB for 5G, 0.06dB for WiFi, 0.26dB for LEO), despite high numbers of consecutive corrupted frames (113, 89, 62, and 150, respectively). This is due to efficient recovery via game states and effective enhancement and inpainting modules. Additionally, received portions of a frame can replace predicted pixels to further improve accuracy.

Comparison with FEC: Table 4 compares the average pixel loss rate and PSNR between VIGOR and oracle FEC, which uses 40%, 35%, 25%, and 70% FEC for 4G, 5G, WiFi, and LEO networks, respectively, to prevent packet loss. Extra FEC transmissions can cause missed deadlines, leading to frame losses. VIGOR achieves higher video quality than FEC, improving PSNR by 0.7-4.7dB with less transmission overhead. Additionally, implementing oracle FEC is impractical due to unpredictable network loss rates, making VIGOR more advantageous.

8.3 End-to-end Latency

We evaluate the end-to-end latency when playing Viking Village, Nature, Corridor, and Viking Village+. Our game server operates on a desktop equipped with an AMD 12-Core Ryzen 9 5900 CPU, Nvidia GeForce RTX 3080 graphics card, and a 2TB SSD. Meanwhile, the client operates on an iPhone12, connected via 4G, 5G, WiFi, and LEO networks. We first evaluate the latency without the need of recovery and the components are as follows:

$$\underset{(30.8-66.6ms)}{T_{non-recovery}} = \underset{(15-42ms)}{RTT} + \underset{(2.9-8ms)}{T_{server\_render}} + \underset{(4.3-5ms)}{T_{server\_encode}} + \underset{(1.6-4ms)}{T_{tx}} + \underset{(7-7.6ms)}{T_{client\_decode}}$$

RTT: This is a Round-trip time (RTT) involved in sending a user's input and receiving the resulting video frame from the server (excluding the transmission time, which is accounted for in T_{tx}). Figure 18 illustrates the CDF of the observed RTTs. The average RTT is $36ms$, $30ms$, $15ms$, and $42ms$ for 4G, 5G, WiFi, and LEO networks, respectively.

T_{server_render} and T_{server_encode}: Upon receiving a user input from the client, the server processes and encodes the video frame using the H.264 codec. Table 5 illustrates that the server's rendering time varies based on the game's complexity, ranging from $2.9 - 8ms$ for the games we evaluated. For a fixed output resolution and encoding parameters, the encoding process takes $4.3 - 5ms$, which has minimal fluctuation.

T_{tx}: The duration required to transmit a frame from the server to the client is influenced by the network bandwidth. Transmitting a 1080p frame takes around $3.6ms$ over 4G, $1.9ms$ over 5G, $1.6ms$ over WiFi, and $4ms$ over LEO networks.

T_{client_decode}: For the games we evaluated, as indicated in Table 5, the client on the iPhone12 processes and decodes an incoming video frame in $7 - 7.6ms$.

End-to-end latency varies with network conditions, system load, and game complexity. If a frame misses its deadline due to delay or loss, VIGOR prompts the client to extract the game state and recover the next frame. As explained in Sec. 7, the end-to-end latency with recovery is as follows:

$$\underset{(<=80ms)}{T_{recovery}} = max(min(\underset{(58ms)}{Timeout}, \underset{(30.8-66.6ms)}{T_{non-recovery}}), \underset{(7ms)}{T_{gs}}) + \underset{(22ms)}{T_{inference}}$$

Given an $80ms$ latency budget and a model inference time of $22ms$, we set the timeout at $58ms$, starting from user input. Video recovery triggers upon timeout or when a corrupted frame is

Table 5. Latency breakdown of server and client's processing time when running different games

Game Name	Server Render	Server Encode	Client Decode
Viking Village	3.6*ms*	4.5*ms*	7.2*ms*
Nature	2.9*ms*	4.3*ms*	7*ms*
Corridor	3.2*ms*	4.8*ms*	7.4*ms*
Viking Village+	8*ms*	5*ms*	7.6*ms*

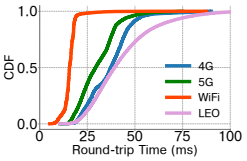


Fig. 18. CDF of Round-trip Time (RTT) for different networks.

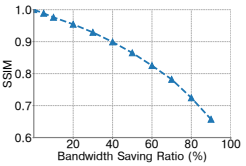


Fig. 19. SSIM degradation with bandwidth saving ratios

detected. Game state extraction runs in parallel with other steps, except model inference. Notably, pixels arriving after timeout but before the deadline can substitute the recovered pixels. This system allows for adjusting the timeout based on different latency requirements, ensuring real-time video recovery. Successfully recovered frames are stored locally and used for potential future recovery.

8.4 Bandwidth Saving

VIGOR cannot only be used to recover video frame loss but also be used to save bandwidth since the server only needs to send certain video frames and the client can reconstruct the remaining video frames based on the game state. As illustrated in Figure 19, the video quality degrades gracefully as the bandwidth saving ratio increases. Note that there is no packet loss in this scenario and SSIM is used as the video quality metric, which ranges between 0 and 1. As it shows, up to 40% bandwidth can be saved while keeping SSIM above 0.9.

8.5 Resource Usage

Viking Village's 1.25 million vertices (200 MB) and Viking Village+'s 630 MB of data fit easily within the GPU's VRAM. According to Unity's documentation [72], PC games typically feature several million vertices, with estimated VRAM usage ranging from 160 MB to 1.2 GB in most modern games. The algorithm works for games with vertex data exceeding 630 MB, and with modern smartphones having shared memory and growing capacities (e.g. 8 GB, 16 GB, 32 GB), our memory requirements remain manageable. With no frame loss, the server renders and transmits all video frames, similar to traditional approaches. In this scenario, iPhone 12's CPU utilization is 35% and energy consumption is 0.05J per frame. For comparison, when the client only receives and decodes video from the cloud server, the iPhone 12's CPU utilization is 33%, also consuming 0.05 J per frame. These results are comparable, suggesting that running the game app on the client side alone has minimal impact on energy consumption. Under 20% frame losses, the corresponding numbers are 41% and 0.06J per frame, and under 100% frame losses, they are 74% and 0.08J per frame. Consequently, in the worst case when every frame needs recovery, the battery life decreases from 10.5 hours to 6.6 hours.

9 CONCLUSION

In this paper, we develop VIGOR, the first real-time deep learning system for reviving cloud gaming sessions. VIGOR efficiently extracts and utilizes game states, generates a mask from H.264 video decoder to indicate which portions of frames need recovery, and develops a novel video recovery model to recover completely lost or partially corrupted frames to cope with server overload, network congestion, and losses in cloud gaming. Our extensive evaluation under diverse network conditions shows VIGOR's effectiveness.

ACKNOWLEDGEMENT

This work is supported by NSF Grant CNS-2212297. We appreciate the insightful feedback from CoNEXT 2024 anonymous reviewers.

References

[1] Acceptable Packet Loss. https://obkio.com/blog/acceptable-packet-loss/#:~:text=Highpacketloss%2Ctypicallyabove,feelslowerandlessresponsive.

[2] S. Aigner and M. Korner. Futuregan: Anticipating the future ¨ frames of video sequences using spatio-temporal 3d convolutions in progressively growing autoencoder GANs. *arXiv:1810.01325*, 2018.

[3] A. Alhilal, T. Braud, B. Han, and P. Hui. Nebula: Reliable low-latency video transmission for mobile cloud gaming. *arXiv preprint arXiv:2201.07738*, 2022.

[4] J. T. Barron. A general and adaptive robust loss function. In *Proc. of CVPR*, 2019.

[5] J. Bulman and P. Garraghan. A cloud gaming framework for dynamic graphical rendering towards achieving distributed game engines. In *HotCloud 20*, 2020.

[6] D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black. A naturalistic open source movie for optical flow evaluation. In A. Fitzgibbon et al. (Eds.), editor, *ECCV*, Part IV, LNCS 7577, pages 611–625. Springer-Verlag, Oct. 2012.

[7] W. Cai, R. Shea, C.-Y. Huang, K.-T. Chen, J. Liu, V. C. Leung, and C.-H. Hsu. A survey on cloud gaming: Future of computer games. *IEEE Access*, 4:7605–7620, 2016.

[8] Q. Cao, S. Zeng, M.-O. Pun, and Y. Chen. Network-level system performance prediction using deep neural networks with cross-layer information. In *Proc. of ICC*, 2020.

[9] D.-Y. Chen and M. El-Zarki. A framework for adaptive residual streaming for single-player cloud gaming. *ACM TOMM*, 15(2s):1–23, 2019.

[10] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, H. Shen, M. Cowan, L. Wang, Y. Hu, L. Ceze, et al. {TVM}: An automated {End-to-End} optimizing compiler for deep learning. In *OSDI*, pages 578–594, 2018.

[11] J. Chen et al. Qava: quota aware video adaptation. In *Proc. of ACM CoNext*, 2012.

[12] Y. Cheng, Z. Zhang, H. Li, A. Arapin, Y. Zhang, Q. Zhang, Y. Liu, K. Du, X. Zhang, F. Y. Yan, et al. {GRACE}:{Loss-Resilient}{Real-Time} video through neural codecs. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 509–531, 2024.

[13] S. Choy, B. Wong, G. Simon, and C. Rosenberg. A hybrid edge-cloud architecture for reducing on-demand gaming latency. *Multimedia systems*, 20:503–519, 2014.

[14] chromeos-apk. https://github.com/vladikoff/chromeos-apk.

[15] Cirp iPhone 15. https://cirpapple.substack.com/p/iphone-15-winners-and-losers.

[16] The worldwide cloud gaming industry is expected to reach $14 billion by 2027. https://www.globenewswire.com/news-release/2022/02/10/2382631/28124/en/The-Worldwide-Cloud-Gaming-Industry-is-Expected-to-Reach-14-Billion-by-2027.html#:~:text=The%20Global\%20Cloud%20Gaming%20Market,is%20based%20on%20cloud%20technology.

[17] Corridor. https://assetstore.unity.com/packages/essentials/tutorial-projects/corridor-lighting-example-33630#description/.

[18] Best Selling Smartphones. https://www.counterpointresearch.com/insights/global-top-10-best-selling-genai-smartphones-q1-2024/.

[19] L. Dai, X. Liu, C. Li, and J. Chen. Awnet: Attentive wavelet network for image isp. In *European Conference on Computer Vision*, pages 185–201. Springer, 2020.

[20] A. Di Domenico, G. Perna, M. Trevisan, L. Vassio, and D. Giordano. A network analysis on cloud gaming: Stadia, geforce now and psnow. *Network*, 1(3):247–260, 2021.

[21] D. Finkel, M. Claypool, S. Jaffe, T. Nguyen, and B. Stephen. Assignment of games to servers in the onlive cloud game system. In *Proc. of NetGames*, 2014.

[22] J.-Y. Franceschi, E. Delasalles, M. Chen, S. Lamprier, and P. Gallinari. Stochastic latent residual video prediction. *arXiv preprint arXiv:2002.09219*, 2020.

[23] H. Fu, H. Yuan, M. Li, Z. Sun, and F. Li. Models and analysis of video streaming end-to-end distortion over lte network. In *2016 IEEE 11th Conference on Industrial Electronics and Applications (ICIEA)*, pages 516–521. IEEE, 2016.

[24] B. Gajera, S. R. Kapil, D. Ziaei, J. Mangalagiri, E. Siegel, and D. Chapman. Ct-scan denoising using a charbonnier loss generative adversarial network. *IEEE Access*, 9:84093–84109, 2021.

[25] Geekbench. https://browser.geekbench.com/.

[26] GeForce Now system requirements. https://www.nvidia.com/en-us/geforce-now/system-reqs/, 2022.

[27] X. Gu, C. Wen, W. Ye, J. Song, and Y. Gao. Seer: Language instructed video prediction with latent diffusion models. *arXiv preprint arXiv:2303.14897*, 2023.

[28] Y. Han, D. Guo, W. Cai, X. Wang, and V. Leung. Virtual machine placement optimization in mobile cloud gaming through qoe-oriented resource competition. *IEEE transactions on cloud computing*, 2020.

[29] Z. He, C. Ge, W. Li, L. Qiu, P. Li, and G. Baig. Optimized live 4k video multicast streaming on commodity wigig devices. In *2024 IEEE 44th International Conference on Distributed Computing Systems (ICDCS)*, pages 1131–1142. IEEE, 2024.

[30] Z. He, Y. Yang, L. Qiu, K. Park, and Y. Yang. Nerve: Real-time neural video recovery and enhancement on mobile devices. *Proceedings of the ACM on Networking*, 2(CoNEXT1):1–19, 2024.

[31] H.-J. Hong, D.-Y. Chen, C.-Y. Huang, K.-T. Chen, and C.-H. Hsu. Qoe aware virtual machine placement for cloud games. In *Proc. of NetGames*, Dec. 2013.

[32] T. Höppe, A. Mehrjou, S. Bauer, D. Nielsen, and A. Dittadi. Diffusion models for video prediction and infilling. *arXiv preprint arXiv:2206.07696*, 2022.

[33] R. Houdaille et al. Shaping http adaptive streams for a better user experience. In *Proc. of ACM MultiSys*, 2012.

[34] T. Huang, R.-X. Zhang, C. Zhou, and L. Sun. Qarc: Video quality aware rate control for real-time video streaming based on deep reinforcement learning. In *Proc. of ACM Multimedia*, pages 1208–1216, 2018.

[35] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. In *Proceedings of the 2014 ACM conference on SIGCOMM*, pages 187–198, 2014.

[36] J. Jiang, V. Sekar, H. Milner, D. Shepherd, I. Stoica, and H. Zhang. Cfa: A practical prediction system for video qoe optimization. In *NSDI*, pages 137–150, 2016.

[37] J. Jiang, V. Sekar, and H. Zhang. Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive. In *Proc. of ACM CoNEXT*, 2012.

[38] J. Kang, S. W. Oh, and S. J. Kim. Error compensation framework for flow-guided video inpainting. In *European Conference on Computer Vision*, pages 375–390. Springer, 2022.

[39] J. Kaur and S. Das. Future frame prediction of a video sequence. *ArXiv*, abs/2009.01689, 2020.

[40] J. Kim, Y. Jung, H. Yeo, J. Ye, and D. Han. Neural-enhanced live streaming: Improving live video ingest via online learning. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 107–125, 2020.

[41] L. Kong, B. Jiang, D. Luo, W. Chu, X. Huang, Y. Tai, C. Wang, and J. Yang. Ifrnet: Intermediate feature refine network for efficient frame interpolation. In *Proc. of CVPR*, pages 1969–1978, 2022.

[42] Y.-H. Kwon and M.-G. Park. Predicting future frames using retrospective cycle gan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1811–1820, 2019.

[43] W.-S. Lai, J.-B. Huang, N. Ahuja, and M.-H. Yang. Fast and accurate image super-resolution with deep laplacian pyramid networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41:2599–2613, 2019.

[44] Z. Lai, Y. C. Hu, Y. Cui, L. Sun, and N. Dai. Furion: Engineering high-quality immersive virtual reality on today's mobile devices. In *Proc. of MobiCom*, 2017.

[45] I. Lee, S. Kim, S. Sathyanarayana, K. Bin, S. Chong, K. Lee, D. Grunwald, and S. Ha. R-fec: Rl-based fec adjustment for better qoe in webrtc. In *Proc. of MM*, 2022.

[46] J. Lee, S. Lee, J. Lee, S. D. Sathyanarayana, H. Lim, J. Lee, X. Zhu, S. Ramakrishnan, D. Grunwald, and K. Lee. Perceive: deep learning-based cellular uplink prediction using real-time scheduling patterns. In *Proc. of Mobisys*, 2020.

[47] K. Lee, D. Chu, E. Cuervo, J. Kopf, Y. Degtyarev, S. Grizan, A. Wolman, and J. Flinn. Outatime: Using speculation to enable low-latency continuous interaction for mobile cloud gaming. In *Proc. of MobiSys*, pages 151–165, 2015.

[48] Y. Li, X. Tang, and W. Cai. Play request dispatching for efficient virtual machine usage in cloud gaming. *IEEE Trans. Circuits Syst. Video Technol.*, 2015.

[49] Y. Liao. Learning to predict end-to-end network performance. *PhD thesis*, 2013. https://hdl.handle.net/2268/136727.

[50] M. Marzolla, S. Ferretti, and G. D'angelo. Dynamic resource provisioning for cloud-based gaming infrastructures. *CIE*, 10(1):1–20, 2012.

[51] Z. Meng, X. Kong, J. Chen, B. Wang, M. Xu, R. Han, H. Liu, V. Arun, H. Hu, and X. Wei. Hairpin: Rethinking packet loss recovery in edge-based interactive video streaming. In *Proc. USENIX NSDI*, 2024.

[52] R. K. Mok et al. QDASH: a qoE-aware DASH system. In *Proc. of ACM MultiSys*, 2012.

[53] A. Narayanan, X. Zhang, R. Zhu, A. Hassan, S. Jin, , X. Zhu, X. Zhang, D. Rybkin, Z. Yang, Z. M. Mao, F. Qian, and Z.-L. Zhang. A variegated look at 5g in the wild: Performance, power, and qoe implications. In *Proc. of SIGCOMM*, 2021.

[54] Nature. https://assetstore.unity.com/packages/3d/environments/nature-starter-kit-2-52977#description/.

[55] S. Oprea, P. Martinez-Gonzalez, A. Garcia-Garcia, J. Castro-Vargas, S. Orts-Escolano, J. Garcia-Rodriguez, and A. Argyros. A review on deep learning techniques for video prediction. *arXiv:2004.05214*, 2020. https://arxiv.org/pdf/2004.05214.pdf.

[56] Z. Qi, J. Yao, C. Zhang, M. Yu, Z. Yang, and H. Guan. Vgris: Virtualized gpu resource isolation and scheduling in cloud gaming. *ACM Trans. ACO*, 2014.

[57] A. Ranjan and M. J. Black. Optical flow estimation using a spatial pyramid network. In *Proc. of CVPR*, pages 4161–4170, 2017.

[58] F. A. Reda, G. Liu, K. J. Shih, R. Kirby, J. Barker, D. Tarjan, A. Tao, and B. Catanzaro. Sdc-net: Video prediction using spatially displaced convolution. In *Proc. of ECCV*, 2018.

[59] M. Rudow, F. Y. Yan, A. Kumar, G. Ananthanarayanan, M. Ellis, and K. Rashmi. Tambur: Efficient loss recovery for videoconferencing via streaming codes. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 953–971, 2023.

[60] A. Sankisa, A. Punjabi, and A. K. Katsaggelos. Video error concealment using deep neural networks. In *2018 25th IEEE ICIP*, pages 380–384. IEEE, 2018.

[61] A. Sankisa, A. Punjabi, and A. K. Katsaggelos. Temporal capsule networks for video motion estimation and error concealment. *Signal, Image and Video Processing*, 14(7):1369–1377, 2020.

[62] M. Semsarzadeh, M. Hemmati, A. Javadtalab, A. Yassine, and S. Shirmohammadi. A video encoding speed-up architecture for cloud gaming. In *Proc. of IEEE Int. Conf. Multimedia Expo Workshops*, 2014.

[63] W. Shi, J. Caballero, F. Huszár, J. Totz, A. P. Aitken, R. Bishop, D. Rueckert, and Z. Wang. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In *Proc. of CVPR*, pages 1874–1883, 2016.

[64] Slashdata 2023. https://www.slashdata.co/post/did-you-know-that-60-of-game-developers-use-game-engines.

[65] Y. Sun et al. Cs2p: Improving video bitrate selection and adaptation with data-driven throughput prediction. In *Prof. of ACM SIGCOMM*, 2016.

[66] Z. Tan, Y. Li, Q. Li, Z. Zhang, Z. Li, and S. Lu. Supporting mobile vr in lte networks: How close are we? In *Proc. MACS*, 2018.

[67] A. Terwilliger, G. Brazil, and X. Liu. Recurrent flow-guided semantic forecasting. In *Proc. of WACV*, 2019.

[68] G. Tian and Y. Liu. Towards agile and smooth video adaptation in dynamic HTTP streaming. In *Proc. of CoNEXT*, pages 109–120. ACM, 2012.

[69] S. Tulyakov, M.-Y. Liu, X. Yang, and J. Kautz. Mocogan: Decomposing motion and content for video generation. In *Proc. of CVPR*, 2018.

[70] Unity. https://unity.com/.

[71] Rendering pipeline introduction. https://docs.unity3d.com/Manual/render-pipelines-overview.html.

[72] Unity Optimizing Graphics Performance. https://docs.unity3d.com/560/Documentation/Manual/OptimizingGraphicsPerformance.html.

[73] Viking Village. https://assetstore.unity.com/packages/essentials/tutorial-projects/viking-village-urp-29140#description/.

[74] C. Vondrick, H. Pirsiavash, and A. Torralba. Generating videos with scene dynamics. In *Proc. of NeurIPS*, 2016.

[75] S. Wang, Y. Liu, and S. Dey. Wireless network aware cloud scheduler for scalable cloud mobile gaming. In *Proc. of ICC*, 2012.

[76] Y. Wang, L. Jiang, M.-H. Yang, L.-J. Li, M. Longand, and L. Fei-Fei. "eidetic 3d lstm: A model for video prediction and beyond. In *Proc. of ICLR*, 2019.

[77] Y. Wang, H. Wu, J. Zhang, Z. Gao, J. Wang, P. S. Yu, and M. Long. PredRNN: A recurrent neural network for spatiotemporal predictive learning, 2021.

[78] J. Wu, Y. Guan, Q. Mao, Y. Cui, Z. Guo, and X. Zhang. Zgaming: Zero-latency 3d cloud gaming by image prediction. In *Proceedings of the ACM SIGCOMM 2023 Conference*, pages 710–723, 2023.

[79] J. Wu, C. Yuen, N.-M. Cheung, J. Chen, and C. W. Chen. Enabling adaptive high-frame-rate video streaming in mobile cloud gaming applications. *IEEE Trans. on Circuits and Systems for Video Technology*, 25(12):1988–2001, 2015.

[80] J. Wu, C. Yuen, N.-M. Cheung, J. Chen, and C. W. Chen. Streaming mobile cloud gaming video over tcp with adaptive source–fec coding. *IEEE Transactions on Circuits and Systems for Video Technology*, 27(1):32–48, 2016.

[81] Z. Xing, Q. Dai, Z. Weng, Z. Wu, and Y.-G. Jiang. Aid: Adapting image2video diffusion models for instruction-guided video prediction, 2024.

[82] D. Xu, A. Zhou, X. Zhang, G. Wang, X. Liu, C. An, Y. Shi, L. Liu, and H. Ma. Understanding operational 5g: A first measurement study on its coverage, performance and energy consumption. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 479–494, 2020.

[83] X. Ye and G.-A. Bilodeau. Stdiff: Spatio-temporal diffusion for continuous stochastic video prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 6666–6674, 2024.

[84] H. Yeo, C. J. Chong, Y. Jung, J. Ye, and D. Han. Nemo: enabling neural-enhanced video streaming on commodity mobile devices. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, pages 1–14, 2020.

[85] H. Yeo, Y. Jung, J. Kim, J. Shin, and D. Han. Neural adaptive content-aware internet video delivery. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, pages 645–661, 2018.

[86] H. Yeo, H. Lim, J. Kim, Y. Jung, J. Ye, and D. Han. Neuroscaler: neural video enhancement at scale. In *Proceedings of the ACM SIGCOMM 2022 Conference*, pages 795–811, 2022.

[87] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli. A control-theoretic approach for dynamic adaptive video streaming over http. In *SIGCOMM*. ACM, 2015.

[88] C. Yu, Y. Xu, B. Liu, and Y. Liu. "can you see me now?" a measurement study of mobile video calls. In *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*, pages 1456–1464. IEEE, 2014.

[89] W. Yu, Y. Lu, S. Easterbrook, and S. Fidler. Efficient and information-preserving future frame prediction and beyond. In *ICLR*, 2020.
[90] W. Yu, Y. Lu, S. M. Easterbrook, and S. Fidler. Crevnet: Conditionally reversible video prediction. *ArXiv*, abs/1910.11577, 2019.
[91] C. Zhang, Z. Qi, J. Yao, M. Yu, and H. Guan. vgasa: Adaptive scheduling algorithm of virtualized gpu resource in cloud gaming. *IEEE Trans. on Parallel Distrib. Syst.*, 2014.
[92] J. Zhang, F. Liu, H. Shao, and G. Wang. An effective error concealment framework for h. 264 decoder based on video scene change detection. In *Fourth International Conference on Image and Graphics (ICIG 2007)*, pages 285–290. IEEE, 2007.
[93] J. Zhang, Y. Wang, M. Long, W. Jianmin, and P. S. Yu. Z-order recurrent neural networks for video prediction. In *Proc. of ICME*, 2019.

Received June 2024; revised September 2024; accepted October 2024