NERVE: Real-Time Neural Video Recovery and Enhancement on Mobile Devices

ZHAOYUAN HE, The University of Texas at Austin, USA
YIFAN YANG, Microsoft Research Asia, China
LILI QIU, The University of Texas at Austin & Microsoft Research Asia, China
KYOUNGJUN PARK, The University of Texas at Austin, USA
YUQING YANG, Microsoft Research Asia, China

As mobile devices become increasingly popular for video streaming, it is crucial to optimize the streaming experience for these devices. Although deep learning-based video enhancement techniques are gaining attention, most of them cannot support real-time enhancement on mobile devices. Additionally, many of these techniques are focused solely on super-resolution and cannot handle partial or complete loss or corruption of video frames, which is common in the Internet and wireless networks.

To overcome these challenges, we present NERVE, a novel approach in this paper. NERVE consists of (i) a novel video frame recovery scheme, (ii) a new super-resolution algorithm, and (iii) an enhancement-aware video bit rate adaptation algorithm. We implement NERVE on an iPhone 12, and it can support 30 frames per second (FPS). We evaluate NERVE in various networks such as 3G, 4G, 5G, and WiFi networks. Our evaluation shows that NERVE enables real-time video recovery and enhancement, and results in 24% - 83% increase in video Quality of Experience (QoE) in our video streaming system.

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

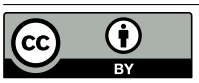
Additional Key Words and Phrases: Mobile video streaming, Video frame recovery, Video enhancement, Bit rate adaptation

ACM Reference Format:
Zhaoyuan He, Yifan Yang, Lili Qiu, Kyoungjun Park, and Yuqing Yang. 2024. NERVE: Real-Time Neural Video Recovery and Enhancement on Mobile Devices. *Proc. ACM Netw.* 2, CoNEXT1, Article 4 (March 2024), 19 pages. https://doi.org/10.1145/3649472

1 INTRODUCTION

Motivation: Machine learning (ML) has seen tremendous progress and penetrated almost every aspect of our lives. One of the major applications of ML is to apply it to enhancing video quality. In particular, videos account for the majority of Internet traffic. However, the Internet bandwidth is highly fluctuating and hard to predict; when the network condition degrades, the video can get lost or its sending rate has to decrease. Many ML-based video enhancement approaches have been developed.

Authors' addresses: Zhaoyuan He, zyhe@utexas.edu, The University of Texas at Austin, 2515 Speedway, Austin, Texas, USA, 78712; Yifan Yang, Microsoft Research Asia, 999 Zixing Road, Minhang District, Shanghai, China, 200241, yifanyang@microsoft.com; Lili Qiu, The University of Texas at Austin & Microsoft Research Asia, Shanghai, China, liliqiu@microsoft.com; Kyoungjun Park, The University of Texas at Austin, Austin, Texas, USA, kyoungjun@utexas.edu; Yuqing Yang, Microsoft Research Asia, Shanghai, China, yuqing.yang@microsoft.com>.



ACM 2834-5509/2024/3-ART4
https://doi.org/10.1145/3649472

Existing ML-based video quality enhancement mostly focuses on improving video resolution given the complete yet lower-resolution video frames. In practice, at the playout time, the receiver may not have a complete video frame to play either due to excessive delay or packet losses. Many measurement studies report a widely fluctuating delay in the Internet and wireless networks (*e.g.*, [31, 53, 60]). Meanwhile, packet losses are also very common. In a wireline network, losses happen due to queue drop during network congestion. In a wireless network, in addition to network congestion and large delay, losses also occur during low signal-to-noise ratio (SNR), collisions, and handoffs. For example, several measurement studies report some receivers experienced loss rates above 10% [9, 60]. [31] reports during handoff the average latency increases by 2.26x and loss rates increase by 2.24x, which significantly degrades video streaming performance. [53] reports the loss rates of 5G sessions are several-fold higher than 4G sessions. Moreover, adding a large buffer may prevent packet drop but lead to the bufferbloat problem, which is prevalent in the Internet (*e.g.*, [17, 18, 25]), causes excessive delay, and harms video streaming performance.

Packet losses not only reduce the available data rate but also lead to a loss of a complete or partial video frame. While Forward Error Correction (FEC) and retransmission have been widely used for loss recovery, their effectiveness is still limited. Retransmission incurs significant delay and may not be acceptable when round-trip time (RTT) is large. FEC is expensive: as we show in Section 3, 35% FEC is required to recover 5% packet losses. Meanwhile, ML-based video prediction can potentially be used to conceal video errors or losses. However, their accuracy is limited since new content will appear in the next video frame, which makes it hard to predict. *Therefore, it is necessary to develop effective video recovery schemes.*

Moreover, various super-resolution (SR) algorithms have been developed to provide good video quality (*e.g.*, [6, 11, 21, 24, 38, 49, 57]). Yet despite significant research, the existing SR cannot support real-time execution on mobile devices. Interestingly, video streaming for mobile devices is becoming increasingly popular. [29] reports that 60%+ U.S. digital video audiences watch videos using their smartphones. *This calls for the development of super-resolution for mobile devices.*

In addition, Adaptive Bit Rate (ABR) has been widely used on the Internet to dynamically adjust the video streaming rate according to the current network conditions. Existing work adjusts the video bit rate based on what content has been received by the client. *As the clients increasingly adopt advanced video enhancement techniques, it is important to use the enhanced video to drive the design of ABR algorithms.*

Our approach: Inspired by the existing video enhancement and its limitation, in this paper we present NERVE by developing (i) a video recovery scheme, (ii) a super-resolution method for mobile devices, and (iii) an enhancement-aware adaptive bit rate algorithm. A nice property of NERVE is that it works with existing video codecs and is easy to deploy.

More specifically, for (i), we observe that simply predicting videos based on previously received data is error-prone. If the sender could send some hints about the current videos in a reliable way, the receiver can use the hints to significantly improve the video prediction. The main challenge is to determine what hint to provide to optimize the recovered video. We develop a novel way to extract compact yet essential states from a video sequence and reliably transmit the states (*e.g.*, using TCP) for video recovery. Inspired by recent quantized image coding techniques [35, 54], we exploit temporal localities in the video and employ edge detection to encode a frame into a very low-resolution binary point code. We find that the learned binary point code encodes both inter-frame movement information and contour information. Our experiments demonstrate that with the learned binary point code, the receiver can significantly improve the quality of video recovery. In our system, we leverage TCP transmission to deliver the binary point code as auxiliary information to help video recovery. Our video recovery has the following distinct advantages: (i)

the binary point code is highly compact: within 1 KB, and yet significantly improves the video quality, (ii) it supports real-time extraction on the server and real-time video recovery on a mobile device, and (iii) it handles both partial and complete video frame recovery.

For (ii), we also advance the state of the art in video enhancement by developing a novel video super-resolution model at multiple resolutions. To address the limited computing power and resources on mobile devices, we use a single neural network for all resolutions and leverage shared parameters to reduce memory overhead. At the bottom of the network, distinct structures are designed to accommodate different resolutions and provide the ABR algorithm with the flexibility to choose different rates. Different from the existing algorithms, it can support (i) different input video resolutions, (ii) real-time execution on mobile devices (*e.g.*, iPhone 12), and (iii) higher video quality of super-resolution.

For (iii), in addition to advancing receiver-side video recovery and enhancement, we develop enhancement-aware video bit rate adaptation. Existing video rate adaptation selects the transmission video rate to maximize the video quality of experience (QoE), which consists of three major factors: video quality determined by the video data rate, change in the video quality, and rebuffering time. The video quality is determined based on the data transmitted to the receiver. Now that the receiver uses various enhancement techniques to improve the video quality, its actual video quality is likely to be much higher than the video directly received from the sender. Therefore, a more effective approach is to use the enhanced video quality to drive the video bit rate decision. We develop an approach to efficiently estimate the impact of video recovery and SR on the video quality and rebuffering time, and use the estimation to facilitate bit rate selection.

To understand the benefit of each component in NERVE, we conduct evaluations using QUIC [22] under a variety of network conditions, including 3G, 4G, 5G, and WiFi networks on iPhone 12. Since iPhone 12 and higher versions account for more than 94% of the total U.S. iPhone purchases in Q1 2023, reported in [12], the performance of most users' smartphones should exceed or be on par with that of the iPhone 12.

Our major contributions can be summarized as follows:

- Our video recovery approach efficiently extracts binary point code from each video frame to better support video recovery at the receiver upon partial or complete video frame losses or excessive delay.
- Our super-resolution method further enhances the resolution of the video frames in real time on mobile devices.
- Our video bit rate adaptation harnesses the full benefit of video recovery and super-resolution approaches by using the video QoE after recovery and enhancement for rate adaptation.
- Our evaluation in diverse types of networks shows that NERVE enables real-time recovery and enhancement and improves the video QoE by 23.7% - 51%, 32.2% - 68%, 37.1% - 83%, and 29% - 72% in 3G, 4G, 5G, and WiFi networks, respectively.

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) video recovery, (ii) video super-resolution, and (iii) ABR algorithms.

Video recovery: Video recovery methods fall into FEC-based and ML-based categories. FEC algorithms, used in systems like DASH and Apple HLS, improve video quality over unreliable networks by adding redundancy to enable lost packet recovery. Popular FEC algorithms include Reed-Solomon (RS) code [37] for correcting burst errors, Low-Density Parity-Check (LDPC) code for efficient XOR-based error recovery, and Convolutional code for correcting errors spread across

multiple bits. The choice of FEC depends on the streaming system's needs and anticipated network errors.

Recent advancements in ML for video prediction include convolutional networks (*e.g.*, [2, 47, 48, 50]), recurrent networks (*e.g.*, [46, 59]), and generative networks (GANs), with GANs being notably effective for generation tasks. Despite their realism in image generation, GANs might not always accurately predict the next video frame. Other approaches use neural networks for video error concealment, allowing for the recovery of corrupted frames using information from previous frames. [40] predicts optical flow from past frames to rebuild damaged sections. [41] employs a capsule network framework to extract and utilize temporal dependencies along with previous frames for motion-compensated error concealment.

Video super-resolution: There have been extensive works on designing video super-resolution (SR) techniques. They leverage previous several video frames to enhance the resolution of the current frame. BasicVSR [6] uses bidirectional RCNNs for feature propagation and PFNL [57] employs a non-local sub-network for pixel correlation within and across frames. Deep learning is commonly applied in motion estimation and compensation (MEMC) methods [11, 21, 24, 38, 49] to improve SR quality through optical flow estimation. NAS [56] offers scalable SR using desktop GPUs for on-demand streaming, whereas NEMO [55] adapts SR to mobile devices with real-time capabilities but requires extensive offline setup. PreSR [61] enhances QoE through selective prefetching but lacks mobile efficiency and incurs high optimization costs. DeepStream [4] combines bitrate optimization with a lightweight SR model but still depends on powerful GPUs. In 360-degree video streaming systems, [8, 13] use SR techniques but do not support general video streaming. Dejavu [15] improves video conferencing visuals by learning from past sessions, but its offline learning approach limits its applicability to dynamic video streaming content.

ABR algorithms: In HTTP-based streaming like DASH and Apple HLS, videos are segmented into chunks lasting 1-10 seconds, encoded at various bitrates. Clients use adaptive bitrate (ABR) algorithms to select appropriate bitrates for each chunk, considering local buffering and download time. A variety of ABR algorithms have been developed [3, 16, 19, 26, 43, 58]. Among these approaches, MPC [58] and Pensieve [26] demonstrate that directly optimizing for the desired QoE objective delivers better outcomes than heuristics-based approaches. Pensieve employs deep reinforcement learning to understand the influence of previous decisions on video quality. While these algorithms adjust for bandwidth fluctuations, they often overlook the effect of client-side video enhancement. NAS [56] and NEMO [55] consider enhancement impacts but lack comprehensive algorithms, and streaming DNN models can introduce extra latency without benefiting all chunks. Unlike these, our method accommodates general video content without video-specific offline DNN training and incorporates the effects of video recovery and SR into its ABR algorithm.

Summary: NERVE advances the state of the art by (i) introducing a novel video recovery scheme that uses compact state information of the current frame for more accurate predictions with minimal overhead; (ii) developing a new video super-resolution method optimized for real-time performance on mobile devices, overcoming the limitations of existing solutions; (iii) adapting video bit rates to optimize QoE after applying video recovery and enhancement; and (iv) showcasing its effectiveness across various network conditions, including 3G, 4G, 5G, and WiFi networks.

3 MOTIVATION

In this section, we conduct controlled experiments to shed light on the limitations of existing FEC, super-resolution, and video ABR algorithms.

Limitations of FEC: FEC has been widely used for video frame recovery. For example, WebRTC is a popular technology that supports streaming videos as well as data and voice on both browsers and

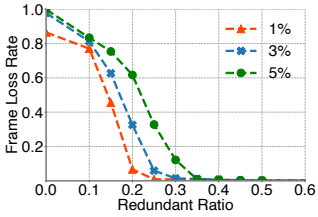


Fig. 1. Frame loss rates under different packet loss rates and FEC redundancy ratios.

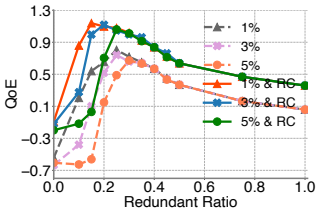


Fig. 2. QoE with different FEC redundant ratios and with or without video recovery (RC).

	Methods			
	RLSP[14]	BasicVSR[7]	CKBG[52]	ours
FLOPS(G)	132.94	71.33	17.8	10.8
params(K)	1154	1887	1750	1619
latency(ms)	5000	3500	1000	22
PSNR	28.5	29.8	29.7	27.1
SSIM	0.814	0.853	0.851	0.801

Table 1. Comparison with other SR methods. FLOPS and latency were validated on the REDS4[30] dataset using 180*320 resolution as input for 4x up-scaling, produced on an iPhone 12.

native clients. It uses a hybrid of Negative Acknowledgement (NACK) and FEC for error recovery during video streaming. When roundtrip time (RTT) is low, NACK is used; otherwise, FEC is used to avoid excessive delay. We evaluate the benefits of FEC. As shown in Figure 1, 1% packet loss requires 25% FEC to achieve close to 0 video frame loss. The corresponding numbers for 3% and 5% packet losses are 30% and 35% FEC, respectively. These numbers show that FEC is very expensive. Similar observations are reported in [23].

Figure 2 further plots the video QoE under different packet loss rates. We make a few observations. For a given packet loss rate, the video QoE first increases with the redundancy ratio as more packets are recovered; then it decreases as the extra redundancy only adds overhead. The redundancy ratio corresponding to the highest QoE is the necessary redundancy required to recover the packet loss. We observe the required FEC redundancy is 25%, 25%, and 30% for packet loss rates of 1%, 3%, and 5%, respectively. The required FEC redundancy is over 6 times the packet loss rate, which is rather expensive. These results motivate our work on developing ML-based video recovery. Combining our video recovery with FEC reduces the required FEC redundancy: the corresponding redundancy reduces to 15%, 20%, and 25%, while achieving higher peak QoE values of 1.15, 1.11, and 1.06 respectively. Moreover, as shown in Figure 2, the QoE under 0 FEC redundancy is much worse than the peak QoE for all curves because removing FEC results in lots of unrecovered losses, which degrade the video performance under no recovery and reduce the number of video frames that are available to use as references for our video recovery.

Limitations of existing super-resolution algorithms: Table 1 compares existing SR algorithms on an iPhone 12 using REDS [30] dataset for Peak Signal to Noise Ratio (PSNR) [34] and Structure Similiarity (SSIM) [44] metrics, highlighting the need for mobile-optimized ML recovery and SR due to the slow performance of current methods. NERVE reduces computational demands to enable real-time performance on mobile devices by using smaller feature maps, efficient feature utilization, and optimized warping. We achieve efficiency by leveraging low-resolution binary point code (64×128), which maintains video quality while simplifying future frame prediction and significantly cutting latency.

4 VIDEO RECOVERY

In this section, we present a deep neural network-based video recovery model for clients to recover videos whenever video frames get lost. This significantly enhances the resilience of video streaming under diverse network conditions. Note that there have been considerable works on video prediction, which predicts the next video frame based on previous frames and can be applied to recover lost videos in our context. However, predicting solely based on the previous information has limited accuracy whenever the video frame has new content, which is common in reality. Our main idea is to send some hints about the next video frame along with the encoded video to facilitate video

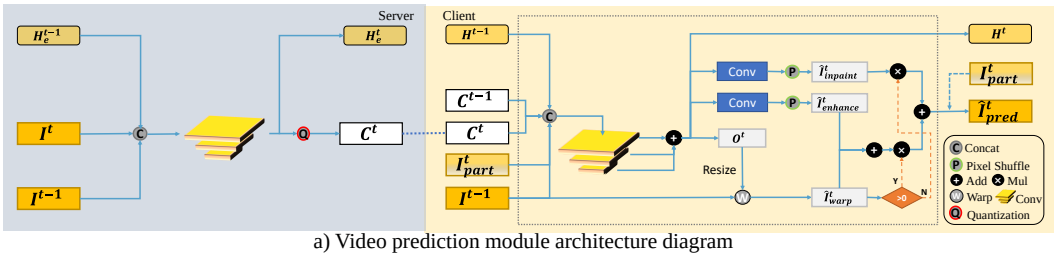
recovery. We design a lightweight mechanism to extract a compact hint for recovering the next video frame and show it can significantly enhance the quality of the recovered videos. Below we describe (i) how to extract the compact binary code at video servers and (ii) how to leverage the code for video recovery at clients.

Extracting binary point code: To extract meaningful information from a video frame to help the video recovery, our encoder borrows the concept of edge detection to preserve the contours of objects. We adopt PidiNET[45] trained on BSDS500[5] as an edge encoder due to its stable performance and small inference overhead. The result of edge detection is usually a value between 0 and 1, so we binarize it to form a binary point code. The resolution of this binary point code can be very low. We find that even a 64×128 code (only 1 KB) can significantly improve the quality of the video recovery. We use TCP to reliably transmit it at a low cost.

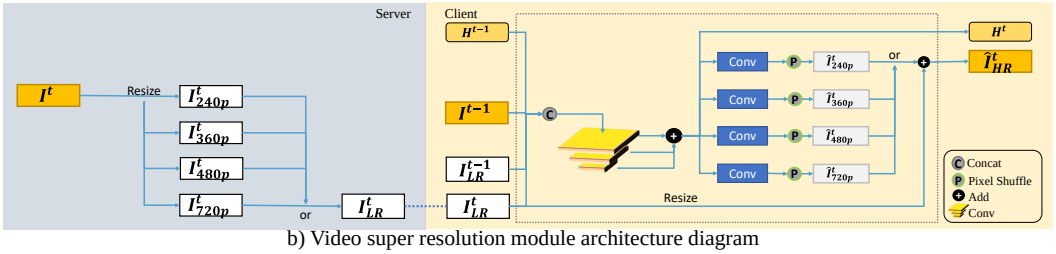
We train the encoder end-to-end with the decoder for better recovery performance. We use a binarization layer in Movement Pruning [39]. To ensure the encoder-decoder structure is differentiable, we skip the gradient from binarization, which directly passes the gradient before the quantization layer to the upper layer.

Figure 6 illustrates the learned binary point code. It captures the motion and contour information of the current video frame. Despite its small size, it significantly improves the prediction accuracy.

Leveraging binary point code: The client recovers the current frame based on the current binary point code, previous video frames, and optionally a portion of the current frame that is received correctly.



a) Video prediction module architecture diagram



b) Video super resolution module architecture diagram

Fig. 3. Model architecture. The blue portion shows the server side operation and the yellow portion shows the client side. I^t and I^{t-1} are the video frames of the current time step t and the previous $t-1$. H_e is the history state maintained by the server-side encoder and H is the history state maintained by the decoder. C is the binary point code generated by the server, which is transmitted to the client to help with video recovery. I^t_{part} is the partially decoded image when the transmission fails. O is the current optical flow. I^t_{240}, I^t_{360}, I^t_{480} and I^t_{720} are different resolutions of video frames downsampled from I^t. I^t_{LR} is the input frame of the super-resolution network received from the server, which is one of the four resolutions mentioned before, and will choose its corresponding upsampling structure to generate high-resolution prediction $\widehat{I}^t_{HR}$.

Our video recovery approach consists of the following steps: (i) estimating the optical flow between two consecutive binary point codes, (ii) upsampling the optical flow to the resolution of

the video frame, and warping the previous video frame to the current one using the optical flow, (iii) further enhancing the content that is warped from the previous frame, and also using inpainting to generate new content that does not appear in the previous frames.

For (i), we use SpyNet [36], an efficient optical flow network, to derive the delta between the two binary codes. Different from SpyNet, we fine-tune the optical flow network end-to-end to warp the previous frame to match the current frame.

For (ii), we find that warping on the iPhone 12 is very slow. We improve its speed by changing the warping resolution to 270p, which reduces the warping time to within 5 ms.

For (iii), we improve the content in warped regions based on the difference to the ground truth. Meanwhile, we observe that warp-based video frame prediction predicts the movement of the content that already exists in the previous video frames. We further use the binary point code from the current video frame to generate new content by upsampling it to the video frame size and using an inpainting module to generate the content in the region that is left empty after warping. It then concatenates the two prediction results to produce the final video frame. We use the Charbonnier loss [6], a widely used loss function for generation tasks, as the optimization metric during training.

To support partial error concealment, we introduce I_{part} as an additional input. I_{part} is a partially decoded video frame due to packet loss, and it is a valuable input for recovering the missing part. We feed it to the recovery model to utilize this partial information, and partial content is also used to override the predicted $\widehat{I}_{pred}$ in the corresponding region. Refer to Figure 9 for I_{part} and the recovery results for the missing region.

Recovery model implementation details: Figure 3(a) shows the overall framework of our video recovery. The server transmits the binary point code to the client to help with video prediction. To improve computational efficiency, all inputs to the optical flow network are downsampled to 64×128. Outside the optical flow network, we resize the output to 270×480 and feed it to subsequent convolution layers. We use PixelShuffle [42] to upsample by 4x to produce 1080p output. Since warping operation on a mobile device is too costly, we resize the 1080p I^{t-1} frame to 270p resolution and then perform warping to generate $\widehat{I}^{t}_{warp}$. However, it is hard to avoid information loss due to downsampling, so this degradation needs to be compensated by the enhancement module. So we additionally feed the 270p I^{t-1} into extra convolution layers to compensate for the loss.

Our video recovery model is trained end-to-end. H_e and H are two hidden layer states maintained at the server and client, respectively. H_e is responsible for capturing the temporal information during the encoding of the binary point code at the server, while H records the temporal information associated with the video recovery process based on the binary point code at the client. During the training phase, at each time step, both H_e and H serve as differentiable information medium that can select valuable features from the historical frames for predicting the current frame. We select a set of feature maps from the output of the optical flow network and apply two groups of convolution layers with non-shared parameters. This results in two intermediate predictions: $\widehat{I}_{inpaint}$ and $\widehat{I}_{enhance}$. The former focuses on filling the gaps in $\widehat{I}_{warp}$ created by the warping process (where the newly emerged content cannot be matched by the optical flow), while the latter concentrates on further adjusting $\widehat{I}_{warp}$. Warping is only based on moving pixels from historical content, while $\widehat{I}_{enhance}$ predicts more subtle changes, such as variations in light or shadow.

Joint FEC and video recovery: So far, we have focused on designing a video recovery model. In practice, one can use both FEC and video recovery to cope with network losses and excessive delay. As shown in Figure 2, the best QoE performance is achieved when we add an appropriate amount of FEC. To determine the right amount of FEC to add to our video recovery, we take five different videos from each category on YouTube (described in Section 8.1) and play them under different network loss rates, where a loss means the packet is either lost or not received in time. For

each network loss rate, we apply different levels of FEC and perform video decoding and recovery as described above. We derive the resulting QoE and select the FEC that yields the highest QoE. In this way, we offline build a lookup table that specifies the best FEC level for each loss rate. During online running, we do not assume perfect packet loss rate prediction. Rather, we predict the packet loss rate using Exponential Weighted Moving Average (EWMA): $L_{curr} = \alpha L_{prev} + (1-\alpha)L$, where $\alpha = 0.3$ in our evaluation. We determine an appropriate FEC redundancy ratio based on the predicted loss rate. A similar approach can be applied to support other recovery methods.

5 VIDEO ENHANCEMENT

We develop a DNN for video enhancement in real time. As shown in Section 3, existing super-resolution methods are either too slow or not accurate. Our SR can achieve good accuracy on mobile devices (*e.g.*, iPhone 12) at 30ms per frame.

Figure 3(b) shows our enhancement process. The network structure of our SR and recovery models are similar. Both are based on an optical flow network to align features, and then use upsampling modules to generate higher-resolution frames. To support streaming at different bit rates, the server will resize the video into different resolutions and transmit the resolution requested by the client as the $\mathbf{I}^t_{LR}$. On the client side, the optical flow network computes the pixel shifting between $\mathbf{I}^t_{LR}$ and $\mathbf{I}^{t-1}_{LR}$. To save memory, SR models with different up-scaling factors share the same optical flow network. To support super-resolution at different resolutions, we use independent convolution layers to learn different degradation patterns. The learning target of our SR model is the gap between the bilinear upsampled $\mathbf{I}^t_{LR}$ and the ground truth $\mathbf{I}^t$. As with the recovery model, we use Charbonnier loss to optimize the above target, and the SR tasks using different up-scaling factors are trained simultaneously.

Our SR model stands out due to its unique properties, such as real-time execution on mobile devices and support for multiple input resolutions. This is achieved through an efficient design that incorporates a shared optical flow network for different up-scaling factors and independent convolution layers tailored to specific degradation patterns. Furthermore, our model provides up-sampling for different resolutions without incurring additional computational costs by resizing the feature maps to predict degradation at various resolutions. This innovative approach allows our model to effectively accommodate devices with diverse computational capabilities, ensuring an optimized video streaming experience across a wide range of devices under various network conditions.

6 ENHANCEMENT-AWARE ABR

We develop an enhancement-aware ABR algorithm. It is built on the ABR in Pensieve [26], but advances Pensieve in two respects: (i) it is an enhancement-aware ABR, which considers the impact of video recovery and super-resolution, and (ii) it incorporates the latest Reinforcement Learning (RL) algorithm – Proximal Policy Optimization (PPO) [33]. Below we focus on how to model the impact of video recovery and super-resolution on video QoE and refer the readers to [33] for detailed descriptions of PPO.

Quality of Experience (QoE): The widely used video Quality of Experience (QoE) is a function of bitrate utility, rebuffering time, and smoothness of selected bitrates, which is defined as follows:

$$\frac{\sum_{n=1}^{N} R_n - \mu \sum_{n=1}^{N} T_n - \sum_{n=1}^{N-1} |R_{n+1} - R_n|}{N}$$

where N is the number of video chunks, R_n and T_n represent the video chunk n's bitrate and rebuffering time resulting from its download, respectively, and μ is the rebuffering penalty. Video recovery and super-resolution impact the QoE in two ways: (1) they improve the video quality and

its corresponding bitrate and (2) their running time also affects the rebuffering time. Below we compute (i) how video recovery affects the video quality, (ii) how SR affects the video quality, (iii) how video recovery affects rebuffering time, and (iv) how SR affects rebuffering time.

Impact of video recovery on video quality: Let us first consider how video recovery affects the video quality. The impact can be quantified based on how many video frames go through video recovery and how much video quality changes after recovery. First, we estimate the number of video frames in the current video chuck that need recovery. We observe that recovery takes place when either (i) a video frame is lost (*e.g.*, due to network congestion, low SNR, mobility) or (ii) a video frame arrives too late (*i.e.*, the frame has not arrived by the time the video frame is scheduled to play). We can predict the loss rate (*e.g.*, using Exponential Weighted Moving window Average (EWMA) or Holt-Winters (HW)), and use the predicted loss rate to estimate (i). To estimate (ii), we compute the expected play time for the i-th video frame as $T_{play} = T_{prev} + i\Delta$, where T_{prev} is the end time of the previous video chuck, i is the video frame index, and Δ is the inter-frame time. We compute the expected arrival time for the i-th frame as $T_{arr} = T_{prev} + \sum_i S_i / tput_{curr}$, where S_i is the total size of data for the i-th frame and $tput_{curr}$ is the predicted throughput for the current video chuck, which can be derived using EWMA or HW. Then for every video frame i in the current chuck, we count the number of frames whose $T^i_{play} < T^i_{arr}$.

Next, we need to compute the video quality for the recovered frame. This depends on the selected video bit rate. For simplicity, we first take videos from the top ten popular categories [27] on YouTube; for each bit rate, we then compute the average PSNR of these video frames after applying video recovery. We use this value as an estimation for the video quality. For the remaining videos that do not need recovery, we can estimate the video quality based on the selected bit rate (*i.e.*, computing the average PSNR of the video frames at that rate).

Impact of SR on video quality: The impact of SR can also be quantified based on (i) the number of video frames that go through SR and (ii) the enhanced video quality after applying SR. (ii) can be estimated by computing PSNR of the video frames after applying SR at each bit rate. Below we compute (i). Since rebuffering time is more annoying than degraded video quality, we skip SR if SR can cause rebuffering. Therefore video frames can be grouped into the following three scenarios: 1) those that are not received in time for playout and require recovery, 2) those that are received in time and can be applied with SR before the playout time, and 3) those that are received in time but cannot finish SR before the playout time. To derive 2), for every video frame i in the current chuck, we count the number of frames whose $T^i_{play} > T^i_{arr} + T_{SR}$, where T_{SR} is the processing time of SR.

Impact of video recovery on rebuffering time: To quantify the video recovery on the rebuffering time, we observe that only the frames that go through recovery have rebuffering time. These frames can be recovered either when they are received successfully but late or corrupted due to network losses. Therefore, their rebuffering time can be estimated as $min(\sum_i S_i / tput_{curr} - T^i_{play}, T_{RC})$, where T_{RC} is the recovery time.

Impact of SR on rebuffering time: As mentioned earlier, to minimize rebuffering time, only the frames that can finish SR before their playout time will go through SR. Therefore, SR does not affect rebuffering time.

Bit rate selection: Our final ABR algorithm computes the QoE of the current video chuck for each video bit rate and selects the rate that leads to the highest QoE.

7 SYSTEM IMPLEMENTATION

We implement and deploy NERVE in a video streaming system, shown in Figure 4.

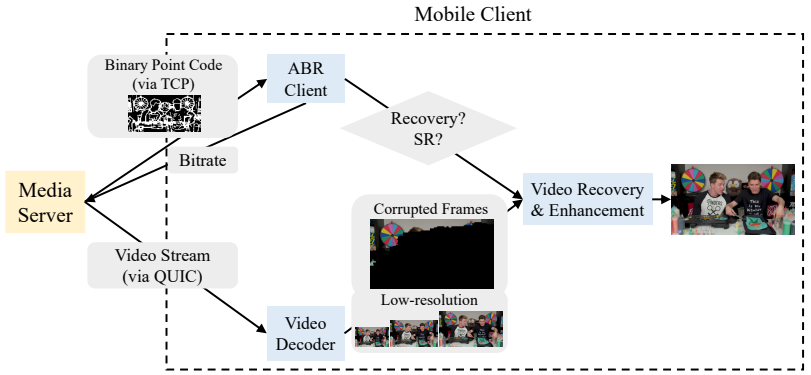


Fig. 4. Overview of our system architecture with NERVE.

Server: The video server encodes videos using standard video codecs like VP9, H264, or H265, and transmits binary point codes to the client via TCP while streaming video content through QUIC [22]. QUIC, favored by platforms like YouTube and Instagram for its efficiency, offers quicker connection setup and faster retransmission than TCP. Google noted QUIC cuts Google search latency by over 2% and YouTube rebuffering time by 9% [10]. Despite its advantages, even with fast retransmission, QUIC experiences a 1.6% packet loss in 5G networks.

Client: The client uses the enhancement-aware ABR algorithm to select and request video bit rates from the server, decodes video frames as usual, and applies the super-resolution (SR) method to enhance resolution before buffering. Incomplete frames are recovered using our video recovery method before playback. We use an iPhone 12 as the mobile client in this paper.

Model deployment: To enhance videos on mobile devices, we deploy our SR and recovery models using CoreML for optimized iOS performance across CPU, GPU, and Neural Engine. Our CoreML model outperforms ONNX, PyTorch Mobile, and TensorFlow Lite in speed. Addressing the slow grid sample operation due to the lack of GPU support, we utilize Metal Performance Shaders (MPS) for a custom, GPU-accelerated layer. By warping at 270p instead of 1080p, warping time decreases from 29*ms* to 5*ms* on the iPhone 12. Utilizing FP16 precision for inputs and weights, we achieve a total inference time of 22*ms* without performance degradation.

8 EVALUATION

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

8.1 Evaluation Methodology

Network traces: We collect network traces from 3G, 4G, 5G, and WiFi networks via QUIC for evaluation, using Chrome's *net-export* [32] for QUIC packet data while watching YouTube videos, and identifying packet loss through the number of retransmitted packets due to loss detection and probe timeout from QUIC logs. Downlink throughput is also measured with iPerf between an Azure server in the central U.S. and a local client. The measurement includes static and walking scenarios for mobile networks and introduces mobility to WiFi traces by moving the client randomly.

Video datasets: For our evaluation, we utilize the video dataset from NEMO [55]. We choose videos from the top ten popular categories [27] on YouTube: 'Product review', 'How-to', 'Vlogs', 'Game play', 'Skit', 'Haul', 'Challenges', 'Favorite', 'Education', and 'Unboxing'. From each category, we select five videos from distinct creators that support 4K at 30fps and are at least 5 minutes long. Then, four of them are allocated to training and the remaining one is used for testing. We transcode them into multiple bit rates ({512, 1024, 1600, 2640, 4400}kbps) at resolutions ({240, 360, 480, 720,

	3G	4G	5G	WiFi
Amount	45	62	53	68
Avg. Duration (s)	322	317	302	309
Avg. Throughput (Mbps)	7.5	21.6	36.4	82.3
Avg. Packet loss rate (%)	0.9	1.3	1.6	0.5

Table 2. Network traces

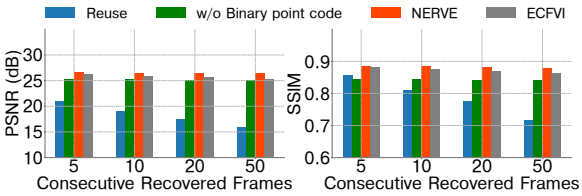


Fig. 5. Performance results of video prediction.

1080}p) using the VP9 codec, following Wowza's guidelines [51], with a GOP size of 120 frames (4 sec).

Performance metrics: We measure the video quality using SSIM and PSNR metrics, where higher values signify better quality. System performance is quantified by QoE. A higher QoE indicates better streaming experience.

Training details: We use 500,000 iterations for all training rounds. We use an 8 GPU Tesla V100 machine with a batch size of 16. We set the learning rate to $1e^{-5}$, and use the cosine decay.

8.2 DNN Performance

First, we evaluate the DNN performance in terms of video quality.

DNN performance of video recovery: Figure 5 compares the video quality of (i) simply reusing the previous video frame, (ii) predicting without the binary point code, (iii) predicting using NERVE with the binary point code, and (iv) predicting using ECFVI [20], a state-of-the-art flow-guided video inpainting model. In (ii), we use the last two consecutive frames to replace the binary point code input in Figure 3. In (iv), ECFVI recovers video frames at 240p resolution. To support a high resolution, we add extra convolutional layers and introduce a pixel shuffle layer to ECFVI to output 1080p video frames. We apply CoreML optimization to ECFVI and test it on an Apple MacBook Air because its large memory requirement precludes its use on an iPhone 12. Running an inference on ECFVI takes 6s on the MacBook Air, while our model takes only 18ms.

We use these schemes to predict the next 5, 10, 20, and 50 frames and calculate the average video quality. Our findings reveal that video recovery without binary point code improves PSNR by 4-9dB and SSIM by 0.03-0.13 over frame reuse. NERVE further enhances PSNR by 6-12dB and SSIM by 0.04-0.17 by incorporating binary point code, demonstrating its efficacy. NERVE also outperforms ECFVI, a much larger model, in both efficiency and accuracy. It improves PSNR by 0.3-1.2dB and improves SSIM by 0.005-0.014. Moreover, as we increase the number of future frames to predict, the prediction quality using our recovery model degrades more gracefully than the other schemes for a similar reason. Figure 6 visualizes our video recovery results, demonstrating our model's ability to learn motions between consecutive frames, with recovered frames closely resembling ground truth. Despite significant differences between consecutive frames, the model generates accurate predictions even in areas without references.

Figure 7 shows partial video recovery performance in a WiFi network, where partially corrupted frames are recovered. Without binary point code, our recovery model improves PSNR by 0.6-5dB and SSIM by 0.01-0.04 compared to reusing previous frames. NERVE further increases PSNR by 4-8.5dB and SSIM by 0.04-0.06 by incorporating binary point code. NERVE outperforms ECFVI by 0.05-0.75dB in PSNR and 0.004-0.008 in SSIM. The benefit of NERVE over ECFVI is reduced because partial recovery decreases the error and the room for improvement. Moreover, the performance gap widens for our video recovery without binary point code versus reusing previous frames, as I_{part} gives the network precise clues for current frame content inference. Similarly, NERVE outperforms other algorithms by a larger margin than shown in Figure 5, due to a stronger learned association between RGB content and binary point code in decoded areas, enhancing prediction in missing

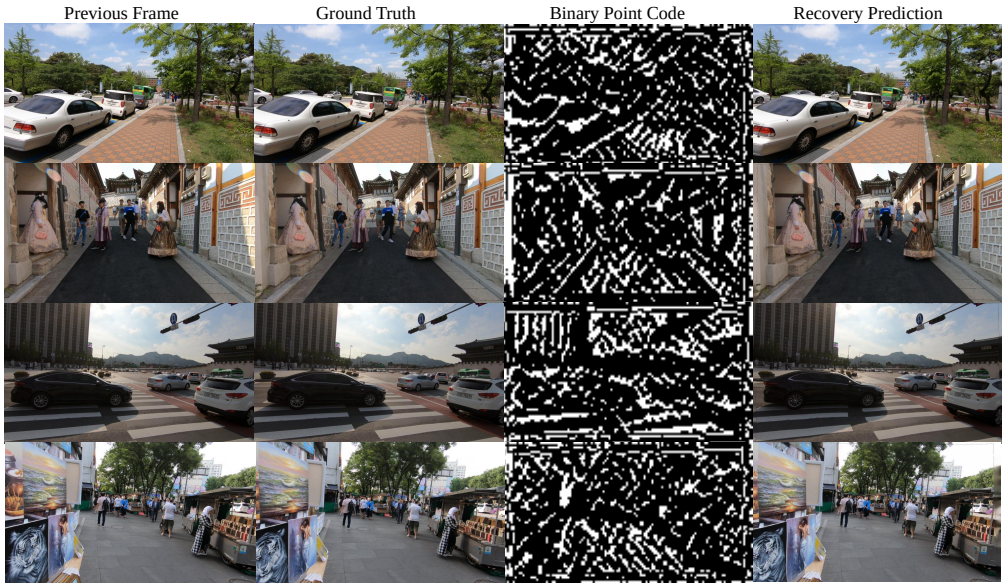


Fig. 6. Visualization of video recovery results. Cases are sampled from REDS4[30].

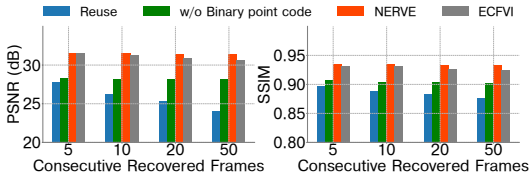


Fig. 7. Performance results of video partial recovery.

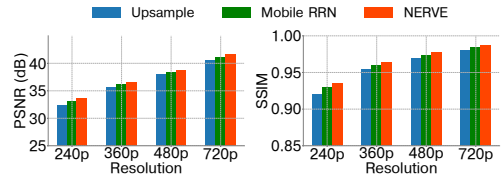


Fig. 8. Performance results of video super-resolution.

sections. Figure 9 plots our error concealment results, showing recovered frames closely resembling the originals, highlighting NERVE's effectiveness in handling both completely lost and partially corrupted frames.

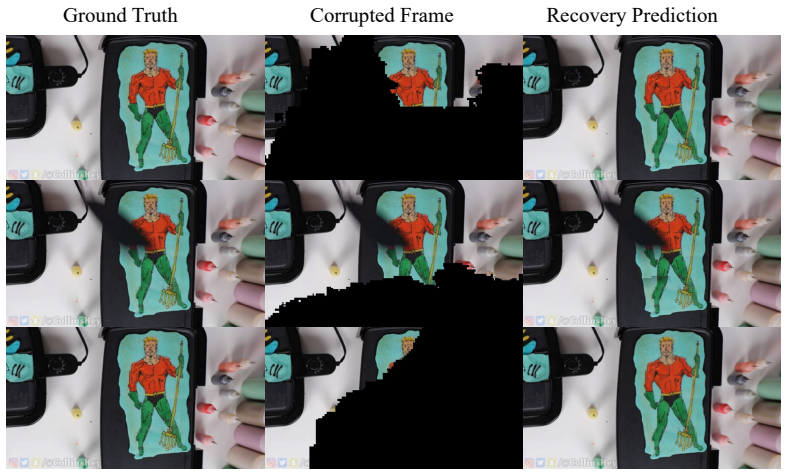


Fig. 9. Visualization of video concealment results. Cases are sampled from REDS4[30].

DNN performance of video super-resolution: Figure 8 shows that our super-resolution (SR) model outperforms Mobile RRN [28] and upsampling on mobile devices. With CoreML optimization,

our SR achieves a 22*ms* inference time on an iPhone 12, faster than Mobile RRN's 28*ms*. NERVE outperforms Mobile RRN in PSNR by 0.4-0.59dB and in SSIM by 0.003-0.006 across 240p to 720p resolutions. Compared to upsampling, NERVE shows greater improvements in PSNR (1-1.3dB) and SSIM (0.007-0.015). Lower-resolution videos yield a higher improvement, as expected. Figure 10 visualizes these super-resolution results, demonstrating our SR's benefit across various resolutions.

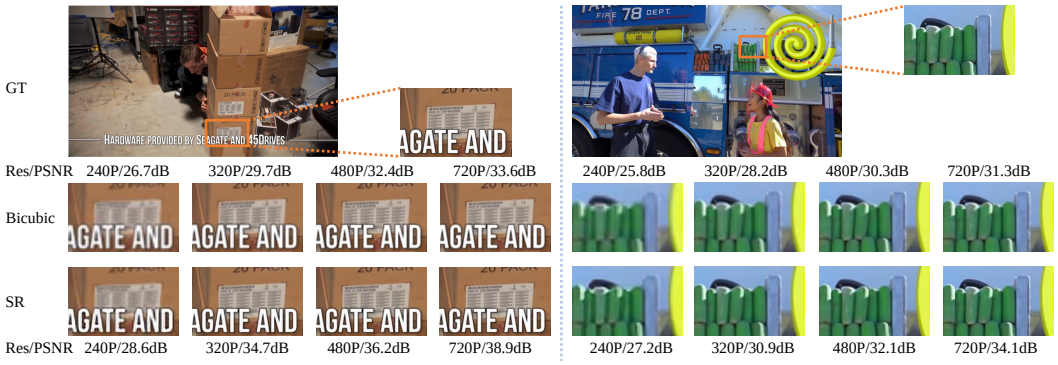


Fig. 10. Visualization of video super-resolution results. Two examples are shown to demonstrate that our super-resolution model achieves significant results at four different scales of video super-resolution.

8.3 System Performance

In this section, we evaluate the system performance in terms of video QoE. Note that we downscale the throughput for all network traces so that their throughput falls into the range between the highest and lowest video bit rates. The average downscaled throughput across all the network traces is around 1-2Mbps.

QoE performance of video recovery: To evaluate the QoE performance of video recovery, we consider three schemes: (i) without recovery model, (ii) RC clone: without recovery-aware ABR, and (iii) NERVE without enabling super-resolution. Note that (ii) means we still perform video recovery for lost or late frames but select the bit rate without taking into account the benefits and cost of video recovery. Figure 11 shows that video recovery alone boosts QoE by 6.3%-14.2% across 3G to WiFi networks due to reduced rebuffering times. Our recovery-aware ABR further enhances QoE by 2.2%-7.5% over recovery alone because it is aware of the usage of recovery for the next frames such that the bit rate can be chosen more wisely to maximize the system QoE. Notably, 5G enjoys the largest improvement because more frames require video recovery as we will show next.

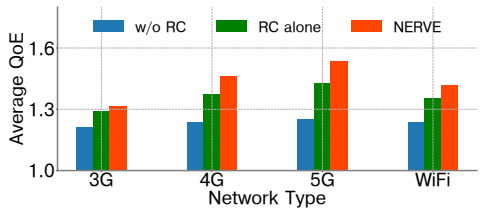
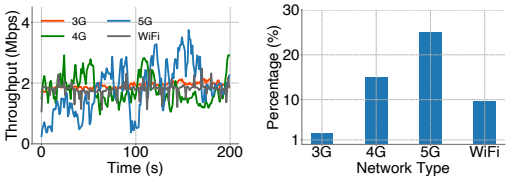


Fig. 11. Qoe of Recovery-only schemes across different network traces.

	3G	4G	5G	WiFi
w/o RC	-0.88	-9.24	-11.86	-1.99
RC alone	0.1	-0.48	-1.21	-0.16
NERVE	0.4	0.07	0.19	0.11

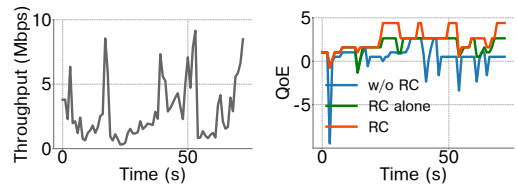
Table 3. QoE of recovered frames

Figure 12(a) illustrates the variability in average throughput across network types, with 5G showing significant fluctuations. Figure 12(b) reveals a high percentage of frames requiring recovery in 5G due to these fluctuations, and even 4G and WiFi networks have around 10% of frames



(a) Average downscaled throughput. (b) Percentage of recovered frames

Fig. 12. Analysis of the QoE performance of video recovery.



(a) Throughput (b) QoE

Fig. 13. Sample time series of throughput and corresponding QoE under 5G network.

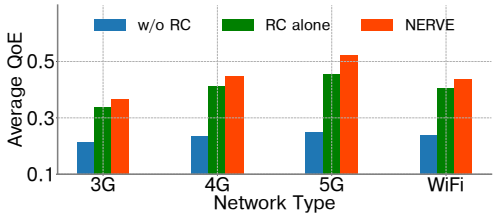


Fig. 14. Qoe of Recovery-only schemes across different network traces.

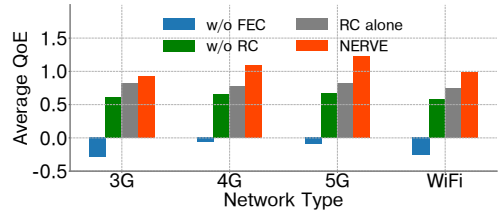


Fig. 15. QoE of Recovery-only schemes with and without FEC under different network traces.

requiring recovery, underscoring the importance of video recovery in unstable network conditions. Figure 13(a) and 13(b) further show a sample time series of throughput and QoE, indicating that QoE suffers greatly without recovery during throughput variations. Recovery alone has a more stable QoE but sometimes gets below 0 due to the rebuffering overhead. Our approach always chooses the bit rate that yields the best QoE. Table 3 shows that video recovery alone boosts QoE for recovered frames by 1.26-10.65, primarily due to decreased rebuffering times. Adding recovery-aware ABR further improves QoE by 0.25-1.4.

QoE performance without FEC under lossy networks: Figure 14 shows the QoE performance of recovery-only schemes across different network traces. Under this setting, we do not enable FEC for loss recovery. For (i), we reuse the last frame when a video frame is late or lost. For (ii) and (iii), our recovery model recovers these frames. In a lossy network, video recovery alone boosts QoE by 58.9%-82.7% across 3G to WiFi networks. NERVE further enhances QoE by 71.8%-110% over without recovery and by 8%-14.6% over recovery alone, highlighting significant improvements, particularly in lossy networks where reusing frames falls short. However, our recovery model can recover many frames with little degradation so its QoE performance is much better.

QoE performance with FEC under lossy networks: Initially excluding FEC, we then jointly optimize FEC and video recovery. Figure 15 compares a baseline without FEC to versions with FEC, where the amount of FEC is determined based on our lookup table. We offline build separate lookup tables that map the packet loss rates to desired FEC levels for different schemes. Our joint optimization enhances performance significantly: 51%-83% over no recovery and 13%-48% over recovery alone across 3G to WiFi networks. It also outperforms the no FEC scenario by 1.2-1.3 in QoE. These results show that (i) FEC plays an important role under lossy network conditions, (ii) the desired amount of FEC depends on the recovery and ABR algorithm, and (iii) each component in NERVE is beneficial.

QoE performance of video super-resolution: Figure 16 compares the QoE of NERVE without enabling recovery with (i) without SR, (ii) SR alone using our model, and (iii) NEMO [55]. NERVE significantly outperforms (i), (ii), and (iii) in QoE: it achieves improvements of 18%-22% over (i), 4.5%-7.1% over (ii), and 0.7%-4.5% over (iii) across 3G, 4G, 5G, and WiFi networks. SR alone improves

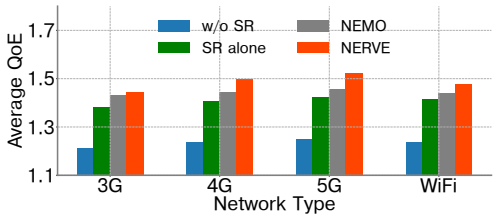


Fig. 16. Qoe of SR-only schemes across different network traces.

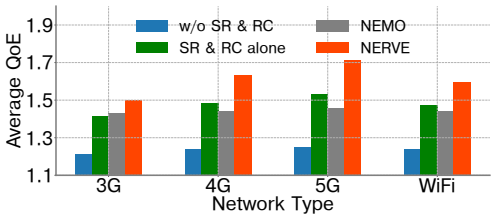


Fig. 17. Qoe of Recovery & SR schemes across different network traces.

QoE by 12%-14%, and SR-aware ABR adds an extra 4%-7% gain, underscoring the value of integrating ABR and SR in NERVE's design.

QoE performance of video recovery and super-resolution: Figure 17 compares the QoE of (i) without SR or recovery, (ii) SR and recovery alone, (iii) NEMO, and (iv) NERVE. NERVE significantly outperforms the alternatives in QoE across 3G, 4G, 5G, and WiFi networks by large margins: 23.7%-37.1% over no SR/recovery, 5.9%-11.9% over SR/recovery alone, and 4.7%-17.4% over NEMO. This demonstrates the substantial impact of both SR and recovery, especially when combined with our enhancement-aware ABR strategy, leading to the highest performance gain. NEMO relies on reusing frames and performs better than other baselines in 3G due to fewer lost/late frames, while NERVE outperforms NEMO by leveraging video recovery and SR across all networks.

Performance of individual components: NERVE enhances QoE greatly by combining video recovery, super-resolution, and FEC. Figure 14 shows that QoEs for 3G, 4G, 5G, and WiFi networks are improved from 0.2-0.24 to 0.36-0.51 by only adding video recovery. Then, adding FEC improves these QoE scores to 0.9-1.24 by improving frame recovery accuracy, shown in Figure 15. As seen in Figure 17, further incorporating SR boosts QoE to 1.5-1.71, demonstrating the strong impact of each component in NERVE on improving video quality across different networks.

8.4 System Latency and Resource Usage

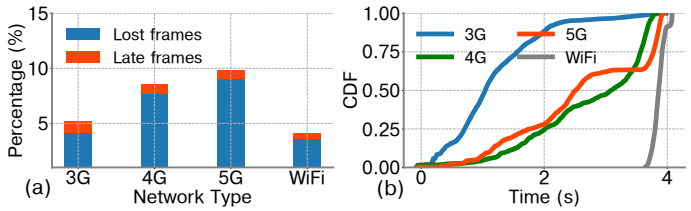


Fig. 18. Analysis of the latency of binary code when the initial buffer size is set to 1 second. (a) Percentage of late and lost video frames. (b) CDF of the time margin (i.e., M) of binary code for all lost and late frames.

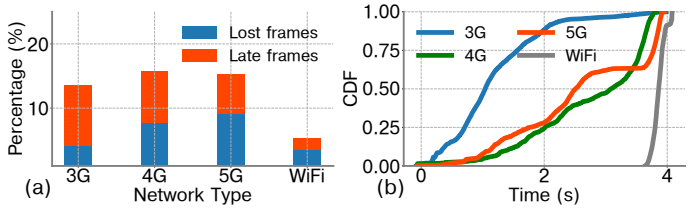


Fig. 19. Analysis of the latency of binary code when the initial buffer size is set to 0.25 seconds. (a) Percentage of late and lost video frames. (b) CDF of the time margin (i.e., M) of binary code for all lost and late frames.

Latency of binary code: The binary code aids video recovery only when its received time plus the inference time is before the deadline (*i.e.*, $M = deadline - inferTime - binaryRxTime \geq 0$), where

the deadline denotes the video frame playout time, $inferTime$ is the time to run our video recovery model, $binaryRxTime$ is the received time of our binary code. Note that the frame playout time is the initial client buffer size plus the inter-frame spacing (i.e., 33ms for 30FPS videos). If UDP packets are retransmitted and decoded in time (*i.e.*, $N = deadline - decodeTime - frameRxTime \geq 0$), the binary code becomes redundant. We set up a video streaming session from an Azure server to an iPhone 12 over 3G, 4G, 5G, and WiFi networks, using NERVE without FEC and SR to recover lost or late frames. The binary code is transmitted over TCP whereas the video data is transmitted over QUIC [1] with built-in retransmission. Across 20 trials per network under both static and walking scenarios, average throughputs are 6.9, 22.3, 34.5, and 78.4 Mbps, respectively.

We first set the initial buffer size to 1s. Figure 18(a) shows 1.0%, 0.8%, 0.7%, 0.5% late frames and 4.2%, 7.7%, 9.1%, 3.6% lost frames for 3G, 4G, 5G, and WiFi networks, respectively. Figure 18(b) plots the CDF of M for all lost and late frames. M is consistently above zero, while video packet loss still happens with QUIC retransmission. The binary code (only 1KB) has a much smaller transmission time and is more reliable. We further reduce the buffer size to 0.25s to understand the impact of buffer size on the latency constraint. As shown in Figure 19(a) and (b), the percentage of late frames is increased to 9.4%, 8%, 6.1%, and 1.7%, and the percentage of $M < 0$ is 0.9%, 0.3%, 0.2%, and 0% for 3G, 4G, 5G, and WiFi networks, respectively. Even though a small percentage of binary code misses the deadline, it is substantially lower than the percentage of lost and late video frames. These results show that even under tight latency constraints, most binary code sent over TCP can still be received in time to help with video recovery.

If the binary code cannot arrive before the deadline, we have the option to recover the frame without the binary code by warping the optical flow estimated using the previous two frames. As described in Sec. 8.2, without binary code, frame recovery using optical flow from two previous frames is less effective than with the binary code but still improves quality significantly with only 22ms inference time on iPhone 12.

System latency: When initiating video streaming, we establish TCP and QUIC sessions. Given that the decoding and model inference can occur simultaneously with the receiving process of subsequent frames, the total latency can be viewed as the sum of the decoding time and the duration required for video enhancement or recovery. The decoding time of 240p, 360p, 480p, 720p, and 1080p videos are 1.8ms, 2.3ms, 2.9ms, 4.1ms, and 6.2ms on the iPhone 12, respectively. Our model adds 22ms for both enhancement and recovery, regardless of the video resolution. This results in a total latency of under 33ms, demonstrating real-time processing capability in our system.

CPU usage and energy consumption: We analyze CPU usage and energy consumption with our recovery and enhancement models. They share a similar model structure and exhibit identical inference time, which implies comparable CPU usage and energy consumption. On an iPhone 12, without DNN processing, CPU utilization is 28% with 0.04J energy per frame. This increases to 37% and 0.05J with 20% frame loss, and 68% and 0.07J with complete frame loss. If every frame goes through recovery or enhancement, the battery life decreases from 13.2 hours to 7.5 hours. The energy consumption can be reduced by optimization techniques, such as quantization and pruning.

9 CONCLUSION

As mobile video streaming becomes increasingly popular, we present NERVE which consists of a video recovery model, a video super-resolution model, and an enhancement-aware bit rate adaptation algorithm. We implement NERVE on an iPhone 12 in a video streaming system and enable real-time neural video recovery and enhancement. Our extensive evaluation shows that NERVE yields significant improvements over other schemes in 3G, 4G, 5G, and WiFi networks.

References

[1] aioquic, 2019. https://github.com/aiortc/aioquic.

[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] Z. Akhtar, Y. S. Nam, R. Govindan, S. Rao, J. Chen, E. Katz-Bassett, B. Ribeiro, J. Zhan, and H. Zhang. Oboe: Auto-tuning video abr algorithms to network conditions. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 44–58, 2018.

[4] H. Amirpour, M. Ghanbari, and C. Timmerer. Deepstream: Video streaming enhancements using compressed deep neural networks. *IEEE Transactions on Circuits and Systems for Video Technology*, 2022.

[5] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik. Contour detection and hierarchical image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(5):898–916, May 2011.

[6] K. C. Chan, X. Wang, K. Yu, C. Dong, and C. C. Loy. Basicvsr: The search for essential components in video super-resolution and beyond. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4947–4956, 2021.

[7] K. C. Chan, X. Wang, K. Yu, C. Dong, and C. C. Loy. Basicvsr: The search for essential components in video super-resolution and beyond. *Computer Vision and Pattern Recognition*, 2021.

[8] J. Chen, M. Hu, Z. Luo, Z. Wang, and D. Wu. Sr360: boosting 360-degree video streaming with super-resolution. In *Proceedings of the 30th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 1–6, 2020.

[9] S. K. Chin and R. Braun. A survey of udp packet loss characteristics. In *Proc. of Conference Record of 35th Asilomar Conference on Signals, Systems and Computers*, 2001.

[10] Chrome is deploying http/3 and ietf quic. https://blog.chromium.org/2020/10/chrome-is-deploying-http3-and-ietf-quic.html.

[11] M. Chu, Y. Xie, J. Mayer, L. Leal-Taixé, and N. Thuerey. Learning temporal coherence via self-supervision for gan-based video generation. *ACM Transactions on Graphics (TOG)*, 39(4):75–1, 2020.

[12] Cirp. https://cirpapple.substack.com/p/iphone-14-pro-and-pro-max-soar.

[13] M. Dasari, A. Bhattacharya, S. Vargas, P. Sahu, A. Balasubramanian, and S. R. Das. Streaming 360-degree videos using super-resolution. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, pages 1977–1986. IEEE, 2020.

[14] D. Fuoli, S. Gu, and R. Timofte. Efficient video super-resolution through recurrent latent space propagation. *arXiv: Image and Video Processing*, 2019.

[15] P. Hu, R. Misra, and S. Katti. Dejavu: Enhancing videoconferencing with prior knowledge. In *Proceedings of the 20th International Workshop on Mobile Computing Systems and Applications*, pages 63–68, 2019.

[16] 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.

[17] H. Jiang, Z. Liu, Y. Wang, K. Lee, and I. Rhee. Understanding bufferbloat in cellular networks. In *Proc. of CellNet*, 2012.

[18] H. Jiang, Y. Wang, K. Lee, and I. Rhee. Tackling bufferbloat in 3g/4g mobile networks. In *Proc. of IMC*, 2012.

[19] J. Jiang, V. Sekar, and H. Zhang. Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, pages 97–108, 2012.

[20] 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.

[21] T. H. Kim, M. S. Sajjadi, M. Hirsch, and B. Scholkopf. Spatio-temporal transformer network for video restoration. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 106–122, 2018.

[22] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar, et al. The quic transport protocol: Design and internet-scale deployment. In *Proceedings of the conference of the ACM special interest group on data communication*, pages 183–196, 2017.

[23] 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.

[24] W. Li, X. Tao, T. Guo, L. Qi, J. Lu, and J. Jia. Mucan: Multi-correspondence aggregation network for video super-resolution. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part X 16*, pages 335–351. Springer, 2020.

[25] J. Lorincz, Z. Klarin, and J. Ožegović. A comprehensive overview of tcp congestion control in 5g networks: Research challenges and future perspectives. *Sensors*, 2021.

[26] H. Mao, R. Netravali, and M. Alizadeh. Neural adaptive video streaming with pensieve. In *Proceedings of the conference of the ACM special interest group on data communication*, pages 197–210, 2017.

[27] Medium report about 'top 10 most popular types of videos on youtube'. https://mag.octoly.com/here-are-the-top-10-most-popular-types-of-videos-on-youtube-4ea1e1a192ac.

[28] Mobile rrn. https://github.com/MediaTek-NeuroPilot/mai22-real-time-video-sr.

[29] Nearly 60% of americans now stream video daily on smartphones, tablets and computers. https://www.nexttv.com/news/nearly-60-of-americans-now-stream-video-daily-on-smart-phones-tablets-and-computers.

[30] S. Nah, S. Baik, S. Hong, G. Moon, S. Son, R. Timofte, and K. M. Lee. Ntire 2019 challenge on video deblurring and super-resolution: Dataset and study. In *CVPR Workshops*, June 2019.

[31] 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.

[32] Capture network log. chrome://net-export/.

[33] Proximal policy optimization (ppo). https://openai.com/blog/openai-baselines-ppo/.

[34] Psnr. https://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio.

[35] A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen, and I. Sutskever. Zero-shot text-to-image generation. *arXiv: Computer Vision and Pattern Recognition*, 2021.

[36] A. Ranjan and M. J. Black. Optical flow estimation using a spatial pyramid network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4161–4170, 2017.

[37] I. S. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal of the society for industrial and applied mathematics*, 8(2):300–304, 1960.

[38] M. S. Sajjadi, R. Vemulapalli, and M. Brown. Frame-recurrent video super-resolution. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6626–6634, 2018.

[39] V. Sanh, T. Wolf, and A. Rush. Movement pruning: Adaptive sparsity by fine-tuning. *Advances in Neural Information Processing Systems*, 33:20378–20389, 2020.

[40] A. Sankisa, A. Punjabi, and A. K. Katsaggelos. Video error concealment using deep neural networks. In *2018 25th IEEE International Conference on Image Processing (ICIP)*, pages 380–384. IEEE, 2018.

[41] 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.

[42] 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 *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1874–1883, 2016.

[43] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman. Bola: Near-optimal bitrate adaptation for online videos. *IEEE/ACM Transactions On Networking*, 28(4):1698–1711, 2020.

[44] Ssim. https://en.wikipedia.org/wiki/Structural_similarity.

[45] Z. Su, W. Liu, Z. Yu, D. Hu, Q. Liao, Q. Tian, M. Pietikäinen, and L. Liu. Pixel difference networks for efficient edge detection. *International Conference on Computer Vision*, 2021.

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

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

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

[49] L. Wang, Y. Guo, Z. Lin, X. Deng, and W. An. Learning for video super-resolution through hr optical flow estimation. In *Computer Vision–ACCV 2018: 14th Asian Conference on Computer Vision, Perth, Australia, December 2–6, 2018, Revised Selected Papers, Part I 14*, pages 514–529. Springer, 2019.

[50] 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.

[51] Wowza's dash bitrate recommendation. https://www.wowza.com/docs/how-to-encode-source-video-for-wowza-streaming-cloud.

[52] J. Xiao, X. Jiang, N. Zheng, H. Yang, Y. Yang, Y. Yang, D. Li, and K.-M. Lam. Online video super-resolution with convolutional kernel bypass graft. 2022.

[53] 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 *Proc. of SIGCOMM*, 2020.

[54] W. Yan, Y. Zhang, P. Abbeel, and A. Srinivas. Videogpt: Video generation using vq-vae and transformers. *arXiv: Computer Vision and Pattern Recognition*, 2021.

[55] 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.

[56] 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.

[57] P. Yi, Z. Wang, K. Jiang, J. Jiang, and J. Ma. Progressive fusion video super-resolution network via exploiting non-local spatio-temporal correlations. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 3106–3115, 2019.
[58] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli. A control-theoretic approach for dynamic adaptive video streaming over http. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 325–338, 2015.
[59] 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.
[60] Y. Zhang, N. Duffield, V. Paxson, and S. Shenker. On the constancy of internet path properties. In *Proc. of IMW*, 2001.
[61] G. Zhou, Z. Luo, M. Hu, and D. Wu. Presr: Neural-enhanced adaptive streaming of vbr-encoded videos with selective prefetching. *IEEE Transactions on Broadcasting*, 2022.

Received June 2023; revised December 2023; accepted January 2024