REPARO: LOSS-RESILIENT GENERATIVE CODEC FOR VIDEO CONFERENCING

Tianhong Li [1] **Vibhaalakshmi Sivaraman** [1] **Pantea Karimi** [1] **Lijie Fan** [1] **Mohammad Alizadeh** [1] **Dina Katabi** [1]

ABSTRACT

Packet loss during video conferencing often results in poor quality and video freezing. Retransmitting lost packets is often impractical due to the need for real-time playback, and using Forward Error Correction (FEC) for packet recovery is challenging due to the unpredictable and bursty nature of Internet losses. Excessive redundancy leads to inefficiency and wasted bandwidth, while insufficient redundancy results in undecodable frames, causing video freezes and quality degradation in subsequent frames.

We introduce Reparo — a loss-resilient video conferencing framework based on generative deep learning models to address these issues. Our approach generates missing information when a frame or part of a frame is lost. This generation is conditioned on the data received thus far, considering the model's understanding of how people and objects appear and interact within the visual realm. Experimental results, using publicly available video conferencing datasets, show that Reparo outperforms state-of-the-art FEC-based video conferencing solutions in terms of both video quality (measured through PSNR, SSIM, and LPIPS) and the occurrence of video freezes.

1 INTRODUCTION

Video conferencing applications are a crucial part of modern life. Despite all the advancements, video conferencing applications still suffer from packet loss, resulting in diminished quality and video freezing (Boyce, 1999; Zheng & Boyce, 2001; Rudow et al., 2023; Karimi et al., 2026). This problem is exacerbated by the strong dependence between encoded frames in traditional video codecs (Bankoski et al., 2011; Mukherjee et al., 2015; Schwarz et al., 2007; Sullivan et al., 2012; av1; Karimi, 2023; Karimi et al., 2024). For instance, in traditional codecs, P-frames (Predicted picture) depend on previous I-frames (Intra-coded picture); hence, the loss of an I-frame affects many subsequent frames, causing a jarring experience when video freezes and subsequently exhibits poor quality until the codec can recover from these dependencies.

Existing systems employ two main techniques to combat this problem: retransmission and forward error correction (FEC). Since real-time applications such as video conferencing must recover lost packets within a limited latency to meet the real-time playback requirement, retransmission is only suitable for scenarios with short round trip times. In all other cases, such applications rely on FEC to recover lost packets within acceptable latency. FEC schemes send redundant packets, known as "parity" packets, to recover the lost data using traditional block codes (Reed & Solomon, 1960; MacKay, 2005) or latency-optimized streaming codes (Rudow et al., 2023). However, all FEC approaches face the challenge of choosing how much redundancy to add, since Internet losses are bursty and unpredictable. Too much redundancy leads to inefficiency and wasted bandwidth, and too little redundancy leads to undecodable frames, causing video freezes and quality degradation in subsequent frames.

This paper presents a novel approach to loss recovery in video conferencing, without using redundant packets or retransmission requests. Instead, when a loss occurs, the receiver leverages the power of "generative" models to reconstruct the missing information. This approach builds upon recent advancements in generative deep learning models (Li et al., 2022; 2023a; Liu et al., 2023; Rombach et al., 2022; He et al., 2022), which are capable of reconstructing images (i.e., frames) even when a significant portion of the data is missing (He et al., 2022; Li et al., 2022). Unlike traditional video codecs, which solely rely on received data for frame reconstruction, generative models operate similarly to humans. They utilize a wealth of knowledge about how people and objects appear, move, and interact to generate the missing information. For instance, when presented with one eye of a person, they can generate the missing eye, or with a partial view of an arm, they can reconstruct the entire torso. The key insight here is that such a generative model is ideally suited for a loss-resilient video codec. However, instead of being guided by textual prompts as seen in typical image generators like DALL-E 2, it can be directed by the received pixels in the current frame, as well as information from past frames, to generate the missing content. This guidance ensures that the generated pixels remain compatible with the correctly

[1]MIT CSAIL, Cambridge, MA, USA. Correspondence to: Tianhong Li <tianhong@mit.edu>.

Proceedings of the 9th MLSys Conference, Bellevue, WA, USA, 2026. Copyright 2026 by the author(s).

received data in the current frame and previous frames, eliminating the potential for hallucinating incompatible content, while generating the data in the lost packets.

We introduce Reparo, a loss-resilient generative codec for video conferencing. The design of Reparo involves two steps. In the initial step, it learns to represent the specific video domain of interest, namely video conferencing, using a small codebook of visual tokens, where each token refers to a patch in a frame. Reparo then operates on these tokens. The transmitter uses a neural network to encode each frame into its respective set of tokens, packetizes them, and transmits the data. Some of these packets may get lost in the network. The Reparo receiver has a neural network that can regenerate the missing tokens from those it receives and its knowledge of how tokens relate to each other in the visual world. Finally, the reconstructed tokens are decoded to produce the original frame. Fig. 1 shows the components of Reparo.

In addition to its resilience against packet loss, Reparo offers three notable advantages:

1. **Efficient Compression**: Reparo efficiently compresses data by capturing prevalent visual features and dependencies among objects and shapes within its codebook. This codebook is pre-learned and known to both the transmitter and receiver, allowing Reparo to transmit only token indices instead of the actual tokens and their underlying dependencies.
2. **Target Bitrate Compliance**: Traditional video codecs exhibit variable bitrates partially due to the significant size discrepancy between I-frames and other frames. This inconsistency makes it challenging for these codecs to meet a precise target bitrate, leading to fluctuations during transmission, transient congestion, and an elevated risk of packet loss or delay. Reparo, in contrast, maintains a constant bitrate as all frames are treated equally, making it easy to adapt to any desired bitrate.
3. **One-Way Communication**: Video conferencing frameworks, including those that rely on FEC (Rudow et al., 2023; web), typically rely on the receiver to send an ACK for every decodable frame. The sender waits for the ACK to retransmit, which could cause even longer freezes when the round-trip time is long. In contrast, in Reparo, the receiver does not need to communicate with the transmitter about undecodable frames; it will always use the received tokens to reconstruct the lost ones.

We have conducted an extensive evaluation of Reparo, comparing it with FEC schemes integrated with WebRTC (web) (ULPFEC and flexFEC), and Tambur (Rudow et al., 2023), a state-of-the-art streaming-code-based FEC approach. The evaluation uses a large corpus of publicly available video clips spanning 5 hours from 84 individuals, which is significantly larger and more diverse than the validation set in prior works on video conferencing (Sivaraman et al., 2022; Rudow et al., 2023; Cheng et al., 2023). The results are as follows:

1. Reparo consistently improves the visual quality of displayed videos across all loss levels. It achieves 33.4 dB, 32.9 dB, and 31.6 dB for the 10% worst PSNR (Peak Signal to Noise Ratio) values under low, medium, and high loss levels, respectively, outperforming state-of-the-art integration of VP9 (a classical video codec)+Tambur by 11.5 dB, 16.4 dB, and 14.7 dB. Notably, this loss resilience does not come at the expense of efficient coding; Reparo achieves similar or better PSNR compared to baselines in the absence of packet loss.
2. Reparo nearly eliminates video freezes and significantly reduces the number of unrendered frames compared to the baselines. Under low, medium, and high loss rates, Reparo fails to render only 0.2%, 0.8%, and 2.0% of frames, while VP9+Tambur fails to render 8.0%, 13.1%, and 29.2% of frames, respectively.
3. In rate-limited environments, Reparo optimally utilizes the full link capacity by consistently transmitting at a fixed desired bitrate. In contrast, VP9+Tambur must maintain a lower average bitrate to prevent packet loss due to the VP9 encoder's bitrate variability. This results in a higher PSNR for Reparo compared to VP9+Tambur (35 dB vs. 33.4 dB).

Our Reparo codec implementation runs in realtime on a V100 GPU, at the transmitter and the receiver, which is in the same performance class as an Apple M2 Max GPU in MacBook Pro laptops (both deliver similar TFLOPS at FP32 precision (NVIDIA Corporation, 2020; Apple Inc., 2023)). Computational requirements are expected to improve over time with the integration of more powerful GPUs into consumer devices.

In summary, video communication has involved a trade-off between efficiency and resilience traditionally. To maximize efficiency, codecs encode frames together (as a delta from a reference frame), whereas to maximize resilience, each frame should be encoded separately. Reparo stands out as the first codec to encode each frame *independently*, with no reliance on other frames, while maintaining efficiency akin to state-of-the-art video conferencing codecs that encode frames together. We believe that Reparo underscores the potential of interdisciplinary design, marrying advances in computer vision with core principles in coding theory and communication systems.

2 Related Work

Video Codecs. Video applications typically use classical codecs such as VP8, VP9, H.264, H.265, and AV1 (Bankoski et al., 2011; Mukherjee et al., 2015; Schwarz et al., 2007; Sullivan et al., 2012; av1). These codecs compress video frames using block-based motion prediction, separating them into keyframes (I-frames) that are compressed independently and

predicted frames (P-/B-frames) that are compressed based on differences between adjacent frames. While classical codecs are widely supported and efficient in slow modes, in real-time video conferencing modes, they are unable to accurately match a desired target bitrate, leading to packet loss and frame corruption when exceeding available capacity (Karimi et al., 2024). Recent generative codecs leverage GANs (Mentzer et al., 2022), multimodal or diffusion-based conditional generation (Yi et al., 2025; Wan et al., 2024), and context-diverse neural compression (Li et al., 2023b), but unlike Reparo, they target rate–distortion improvement rather than loss recovery.

To overcome some of these limitations, several neural codecs have been proposed in recent years (Dasari et al., 2022; Yeo et al., 2018; Khani et al., 2021; Wang et al., 2021a; Lu et al., 2019). These codecs use a low-quality video that is then enhanced using a Deep Neural Network (DNN). Reparo differs from such prior neural codecs in two ways. First, these neural codecs still rely on temporal dependencies, wherein an undecodable frame can cause one or more subsequent frames to freeze. In contrast, Reparo has no dependencies between encoded frames and hence the impact of a loss in one frame does not propagate to other frames. Second, none of the neural approaches use generative neural models that synthesize images from a few small pre-computed tokens.

Loss-Resilient Video Codecs. Beyond coding schemes, system-level approaches improve resilience by designing keypoint-based ultra-low-bitrate conferencing (Zhang et al., 2025) or transport protocols that exploit neural codecs' tolerance to loss (Xia et al., 2025). Forward Error Correction (FEC) is a technique used in communication systems to recover lost data packets without retransmission. Instead of retransmission, redundant information sent by the sender is used by the receiver to reconstruct the original data. This is particularly important in real-time communication systems such as video conferencing, where retransmission of lost packets can cause unacceptable delays. Traditional FEC codes such as parity codes (Begen, 2010), Reed-Solomon (RS) codes (Reed & Solomon, 1960), and fountain codes (MacKay, 2005) are all block codes that are optimal for random losses, where packets are lost independently. Recently, researchers have proposed using streaming codes for FEC (e.g., Tambur (Rudow et al., 2023)), achieving better loss recovery than block codes for bursty losses, where several packets over one or more consecutive frames are lost.

GRACE (Cheng et al., 2023) proposed a neural video codec that can tolerate packet loss. However, there are two key differences with our approach. First, GRACE does not include an explicit loss recovery network. It improves the loss-resilience of an existing neural decoder (DVC (Lu et al., 2019)) by randomly masking information (i.e., using dropout) during its training. Unlike our loss recovery network, GRACE's decoder cannot leverage multiple received frames to reconstruct the missing information. Second, the DVC code underlying GRACE is based on Delta coding which creates temporal dependencies between encoded frames. Therefore, errors in one decoded frame due to packet loss propagate to subsequent frames even if they incur no loss. On the other hand, Reparo encodes each frame independently, and thus does not have error propagation.

Generative Neural Networks. In recent years, there has been significant progress in the development of generative models, which can create text, audio, images, and videos indistinguishable from those created by humans (Devlin et al., 2018; Liu et al., 2023; Rombach et al., 2022; Li et al., 2022). These models use knowledge of the target domain to generate content under certain conditions. For example, a text generative model can generate a paragraph conditioning on text prompts (Devlin et al., 2018), and an image generative model can produce an image using only a partial view (Li et al., 2022).

To enhance the use of domain knowledge, many recent visual generative models have adopted a two-stage design (van den Oord et al., 2017; Razavi et al., 2019; Chang et al., 2022; Yu et al., 2021; Lee et al., 2022). First, they learn to represent the target domain using a visual token codebook. Each visual token corresponds to a patch in the image and the codebook serves as a high-level abstraction of the visual world. Generation is then performed in this token space, similar to text generative models. These models have shown impressive performance in image generative tasks, such as text-to-image synthesis (Rombach et al., 2022; Chang et al., 2023) and image editing (Li et al., 2022).

Given these capabilities, generative models are well-suited for loss-resilient video conferencing. Our work is the first to apply such advances to synthesize video conferencing frames when packet losses occur. By conditioning on (i.e., prompting with) the received data, our method can generate video frames identical to the original frames, achieving loss-resilient video conferencing.

Earlier work also explored using generative decoders directly for compression (Santurkar et al., 2017), foreshadowing our use of generative models for loss recovery.

3 Reparo Design

3.1 Overview

Reparo is a generative loss-resilient video codec specifically designed for video conferencing. As shown in Fig. 1, Reparo consists of five parts: (1) an *encoder* that encodes the RGB video frame into a set of tokens, (2) a *packetizer* that organizes the tokens into a sequence of packets, (3) a *bitrate controller* that adaptively drops some fraction of the packetized tokens to achieve a target bitrate, (4) a *loss recovery module* that recovers the missing tokens in a frame

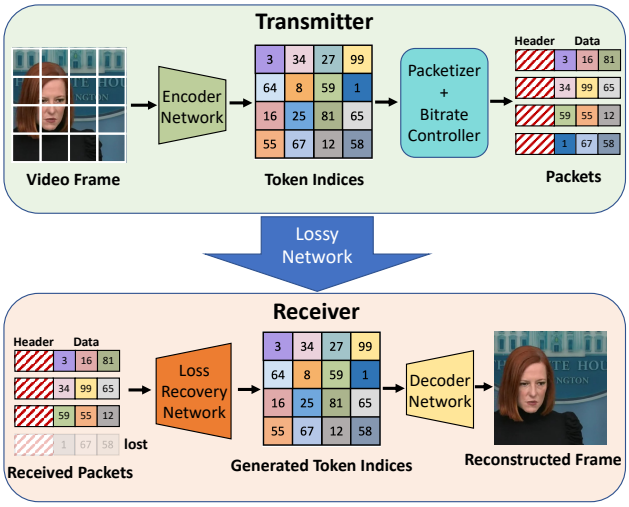

Figure 1: Overview of Reparo. It comprises an encoder-decoder pair responsible for converting RGB frames into quantized tokens and vice versa, as well as new modules for packetization, bitrate control, and loss recovery that operate in the token space.

based on the tokens received by the frame deadline, and (5) a *decoder* that maps the tokens back into an RGB frame. We call the encoder-decoder combination in Reparo its neural codec, while the rest of the components help with loss recovery atop the codec. The encoder, packetizer, and bitrate controller are situated at the transmitter side, while the loss recovery module and decoder operate at the receiver side. We describe these modules in detail below.

3.2 Reparo Components

3.2.1 The Neural Codec: Encoder and Decoder

In contrast to prior work on loss-resilient video conferencing, which utilizes traditional codecs with FEC-based wrappers, Reparo employs its own codec based on the concept of a tokenizer. Tokenizers are commonly used in generative models to represent images using a learned codebook of tokens. Instead of generating images pixel by pixel, images are divided into patches, and each patch's features are mapped to a specific token in the codebook. This reduces the search space of generative models since the number of tokens in an image is much smaller than the number of pixels. Each token represents a vector in feature space. By training a neural network to identify a small number of feature vectors that can best generate all images in the training dataset, a set of tokens is selected for the codebook.

We observe that tokenizers naturally fit the requirements of a codec since they allow us to compress frames in a video by expressing them as a set of tokens, which can be transmitted as indices without the need to transmit the actual tokens. Since the transmitter and receiver share a codebook, the receiver can recover the original frames by looking up the token indices in its codebook and decoding them to the

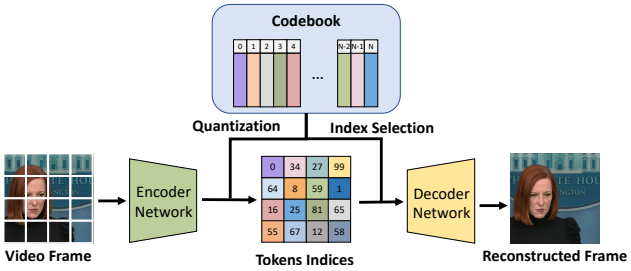

Figure 2: Token-based neural codec. The encoder converts patches from video frames into features and uses a codebook to quantize the features into tokens by finding the nearest neighbor of each feature in the codebook. The decoder then uses the tokens to reconstruct the video frame.

original frames. Further, since each frame is compressed independently of other frames based only on its own token indices, losses in one frame do not affect other frames.

We use a tokenizer called VQGAN (Esser et al., 2021), which consists of an encoder, a decoder, and a codebook (see Fig. 2). The encoder is a convolutional neural network (CNN) that takes patches in an image and maps each one of them to the nearest neighbor vector in the codebook, i.e., the nearest token. The decoder is also a CNN that takes a concatenation of tokens that represent an image and reproduces the original image.

The compression achieved by VQGAN depends on two of its parameters: the number of tokens used for each frame, and the size of the codebook. Since the image is divided into patches, each mapped to a token, the number of tokens dictates the size of each patch within an image. As the number of tokens is increased, the smaller each patch becomes. More tokens allow a more fine-grained reconstruction as it is easier for a token to represent a smaller patch. However, since we transmit token indices from the sender to the receiver, more tokens means more bits for transmitting all of their indices, and reduces the compression factor. Similarly, a larger codebook enables a more diverse set of features to choose from for each token, but requires more bits to represent each token index. Thus, both of these parameters lead to different tradeoffs for the achieved bitrate and visual quality. We show this in Fig. 12.

3.2.2 The Packetizer

After encoding the original image into tokens, Reparo divides them into several packets for transmission. The packetization strategy is designed to avoid placing adjacent tokens in the same packet since the closest tokens in the image space are the most helpful for recovery when a token is lost.

In Fig. 3, the first step in the green box labeled Transmitter shows our token wrapping strategy for an example with 4×4 tokens that are split into 4 packets. The packet index of the token at position (i,j) is $2\cdot(i \mod 2)+j \mod 2$. Tokens

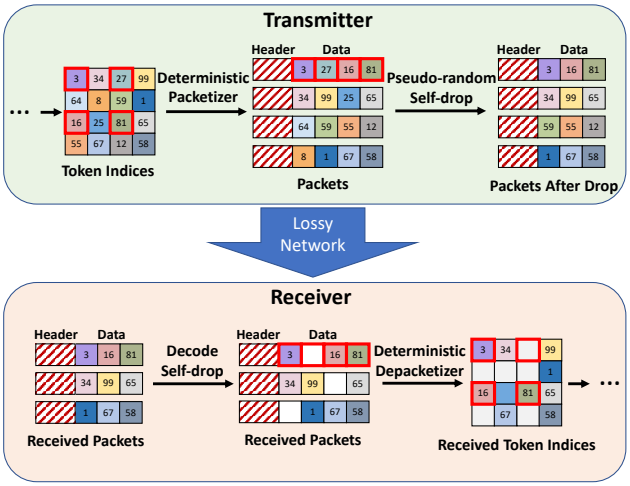

Figure 3: The transmitter first uses a deterministic packetizer to wrap image tokens into packets. Then a bitrate controller drops some tokens in each packet to adapt to the target bitrate. The receiver first decodes which tokens are dropped by the bitrate controller. It then depacketizes the received packets to extract the received token indices with the lost tokens identified.

in each packet are ordered first by their row index, then by their column index in ascending order. This is one of many ways to wrap tokens into packets while avoiding placing adjacent tokens in the same packet. The strategy needs to be deterministic so that the receiver can place the received tokens in the appropriate position in the frame before trying to recover the missing tokens. Each packet has a header including its frame index, packet index, and packet size so the receiver can identify which frame the tokens belong to and how many packets that particular frame has.

3.2.3 *The Bitrate Controller*

Video conferencing applications often need to adjust their bitrate in response to network congestion. In prior work, this was achieved by altering the extent of compression to meet the desired bitrate. In contrast, Reparo can easily adapt its bitrate by dropping tokens, as it is highly resilient to lost tokens and degrades gracefully with increasing loss rates. We call this "self-dropping" since Reparo chooses to drop tokens on its own even before transmitting them. Remarkably, Reparo can tolerate up to 50% token loss with only a minimal impact on video PSNR, as demonstrated in Fig. 5. In practice, Reparo chooses the tokens it drops deterministically based on the frame index and packet index (Fig. 3 top row right). This is to ensure that the receiver can easily identify which token locations were dropped based simply on the frame and packet index in the received packet's header. With this information, the receiver can decode (Fig. 3 bottom row left) the locations of the tokens removed by the bitrate controller.

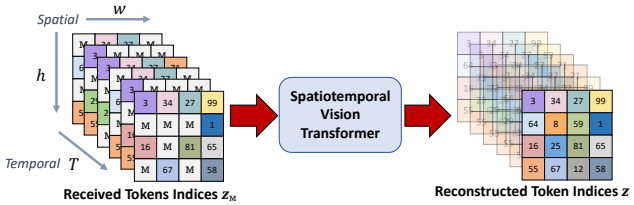

Figure 4: Loss recovery module. It uses a neural architecture based on a spatio-temporal vision transformer to generate any lost tokens using the learned knowledge of how people and objects look, along with the received tokens in the current and recent frames.

3.2.4 *Loss Recovery Module*

The key ingredient for Reparo to carry out loss recovery is a deep generative model that leverages received tokens and video conferencing domain knowledge to generate lost tokens. For instance, the generative model can synthesize all tokens associated with a particular human face based on a subset of those tokens. Similarly, it can produce the token corresponding to a moving hand conditioned on the tokens from previous frames. In the following sections, we provide a comprehensive description of the architecture, training procedure, and inference algorithm of our loss recovery module.

Network Architecture. The loss recovery module is a neural network. It takes as input the received tokens organized according to their positions in the original frame. Lost tokens are expressed with a special token called the Mask token, `[M]`, as shown in Fig. 4. It also takes as input the tokens from the past T frames, which provide the context for the scene.

We use a common neural network architecture called Vision Transformer (ViT) (Dosovitskiy et al., 2021). Transformers have gained widespread popularity in computer vision and natural language processing for predicting missing image patches or words (Devlin et al., 2018; Dosovitskiy et al., 2021; He et al., 2022). The ViT employs an attention module in each layer to aggregate information from all tokens in an image. To predict a missing token, the attention module uses the received tokens and weighs them by their relevance to the missing token. The relevance is computed by performing a softmax over the dot product of each token with every other token. To extend the standard vision transformer structure to video clips, we use a spatio-temporal attention module (Arnab et al., 2021). In each transformer block, we perform attention over the time dimension T (across adjacent frames) and then over the space dimension $h \times w$ within a frame. This enables our loss recovery module to exploit both spatial information from the same frame and temporal information across consecutive frames. Specifically, to generate a missing token, the module can use the nearby tokens in both space and across frames, as those tokens have a strong correlation with the missing token. Performing attention over time and space sequentially significantly reduces the computational cost: attention over

both space and time simultaneously requires $O(T^2h^2w^2)$ of GPU memory, while attention first over time and then over space requires only $O(Th^2w^2+T^2hw)$ of GPU memory.

Leveraging temporal information incurs some overhead as the last few frames need to be held in memory to decode the next frame. Hence, we limit the temporal dependency to a maximum of 6 frames. It is worth noting that using tokens from previous frames for loss recovery does not cause Reparo to stall like traditional codecs due to undecodable frames. Specifically, the spatio-temporal ViT utilizes the six previous frames while decoding the current frame, allowing reuse of received tokens across frames to achieve a better bitrate and loss rate. Every frame is generated and decoded regardless of the previous frame's generation result and based solely on the actually received tokens of previous frames. If more tokens are lost in the previous frames, the quality of the current frame's generation may be poorer, but Reparo will never stop generating or decoding, unlike classical codecs. We provide detailed information about our spatio-temporal ViT structure in §B.2.

It is worth highlighting the difference between our loss resilience and all past work. Traditionally, loss resilience is achieved by encoding frames together and adding FEC, at the transmitter. In contrast, our generative approach allows frames to be encoded *independently* at the transmitter without FEC. The receiver however decodes each frame holistically, looking both at its received tokens and tokens from past frames to generate the missing tokens.

Training the Network. The goal of the training is to ensure the resulting neural network can recover from both network packet losses and tokens self-dropped by the bitrate controller to achieve a particular target bitrate.

Thus, during training, we simulate both types of losses and optimize the network weights to recover the original tokens. Specifically, we simulate the packetization process, and in each iteration, we randomly sample a self-drop ratio r_d from 0 to 0.6. Based on r_d, a certain fraction of tokens are dropped from each packet. Then, a packet drop rate r_p is randomly selected from 0 to 0.8, and packets (and all their tokens) are dropped based on the selected packet drop rate. At the receiver, the tokens that have been received are identified based on frame and packet indices. The missing tokens, whether dropped due to self-drops or packet loss, are replaced with a learnable mask token `[M]` (Fig. 4). This ensures that the input sequence length to the model is fixed regardless of the number of dropped tokens, which is a requirement for ViT. The resulting tokens combined with positional embeddings that provide spatial and temporal location information for each token (including the mask tokens) are then provided as the input of the ViT module. The output of the ViT module is a complete $h \times w \times T$ grid with generated or original tokens in their proper positions (where T represents the number of past frames), but we only use the last frame's tokens to reconstruct the original frame using the codec decoder. Below we describe the loss function used in the training.

Reconstructive Training Loss. Let $z = [z_{ijk}]_{i=1,j=1,k=1}^{h,w,T}$ denote the latent tokens from the encoder, and $M = [m_{ijk}]_{i=1,j=1,k=1}^{h,w,T}$ denotes a corresponding binary mask indicating which tokens are missing in the last T frames. The objective of the training is to reconstruct the missing tokens from the available tokens. To accomplish this, we add a cross-entropy loss between the ground-truth one-hot tokens and the output of the loss recovery network. Specifically,

$$\mathcal{L}_{reconstructive} = -\mathbb{E}z\Big(\sum_{\forall i,j,k,m_{ijk}=1,k=T} \log p(z_{ijk}|z_M)\Big), \tag{1}$$

where z_M represents the subset of received tokens in z, and $p(z_{ijk}|z_M)$ is the probability distribution over the codebook for position (i,j) in the k-th frame predicted by the reconstruction network, conditioned on the input received tokens z_M. As is common practice, we only optimize this loss on the missing tokens of the last frame. Optimizing the loss on all tokens reduces reconstruction performance, as previously observed (He et al., 2022). Detailed training schemes are included in §B.

Inference Routine As the deadline for displaying each frame is hit every 33 ms for 30 fps, we aggregate all received packets for the current frame (as identified by the frame and packet indices) and regard all unreceived packets as lost. Once we place the received tokens in their respective positions corresponding to $h \times w$ patches in the frame, we can determine the exact locations of the missing tokens. We use all the tokens *received* from the previous $T = 6$ frames to perform spatio-temporal loss recovery.

For each token position (i,j) in the current frame, we use $p(z_{ij}|z^M)$, the probability distribution of the predicted token given the received tokens, to choose the token with the highest probability as the reconstructed token. The resulting grid of reconstructed tokens is fed into the neural decoder to generate the RGB frame for display.

4 EVALUATION

We evaluate Reparo and compare to several baselines. We describe the baselines and experimental setup in §4.1. We evaluate baselines and Reparo under network scenarios with random packet loss in §4.2, and under packet losses induced by a rate-limited bottleneck link in §A.2. We discuss Reparo's parameter choices, latency overheads, and qualitative results in §A.3.

4.1 Experiment Setup

Baselines. **ULPFEC** and **flexFEC** are two solutions included in WebRTC (web) to recover from audio and

video packet loss. **Tambur** (Rudow et al., 2023) is a recent streaming-codes based FEC solution atop the VP9 video codec (Mukherjee et al., 2015) that has been shown to perform better than classical block-based FEC techniques. Following the original paper, Tambur is implemented in the Ringmaster video conferencing platform (rin). We set Tambur's latency deadline τ to 3 frames, and the bandwidth overhead of all baselines to be approximately 50%. The frame rate is set to $30\,\mathrm{fps}$ and the video resolution to 512×512, which is typical for video conferencing.

Datasets. To train the neural codec in Reparo, we use three datasets: FFHQ (70,000 images) (Karras et al., 2019), CelebAHQ (30,000 images) (Karras et al., 2017), and a subset of TalkingHeads ($\sim$25 hours of video) (Wang et al., 2021b). These datasets offer high-resolution human face images and videos, well suited for training Reparo, which targets enhanced video conferencing quality. The loss-recovery module is trained exclusively on TalkingHeads, as it requires video clips rather than still images.

For evaluation, we construct a comprehensive video conferencing validation set by combining the Gemino dataset (Sivaraman et al., 2022) with part of TalkingHeads (Wang et al., 2021b), yielding 5 hours of data across 412 video clips and 84 subjects. We ensure no overlap with the training set. This validation set is substantially larger and more diverse than those used in prior work—Gemino (Sivaraman et al., 2022) (25 videos, 5 subjects, 75 minutes), Tambur (Rudow et al., 2023) (20 videos, 200 minutes), and GRACE (Cheng et al., 2023) (60 videos, 745 seconds)—highlighting the robustness and generalization ability of Reparo.

Implementation. We studied two FEC algorithms, ULPFEC and flexFEC, implemented in Google's implementation of WebRTC (web) with VP8 codec. The FEC rate is set to 50% with packet retransmissions disabled, and the frame wait-time is set to $150\,\mathrm{ms}$. This wait-time parameter dictates the duration after which WebRTC stops to decode a frame that has been partially received. Our experimental setup includes a headless real-time video application, built on the WebRTC platform, which processes video input at a rate of $30\,\mathrm{fps}$ at the sender's end and records the output to a file at the receiver's end. The network conditions between the sender and receiver were simulated using the Mahimahi network emulator (Netravali et al., 2015). Within this environment, we constructed a loss shell akin to that used in Tambur, facilitating controlled network loss scenarios for our experiments. Tambur is evaluated using the Ringmaster platform with VP9 codec.

Our neural codec and loss recovery modules are implemented in PyTorch, and they operate in real-time on two V100 GPUs, processing 30 fps 512x512 videos. One GPU was used for the transmitter and one GPU for the receiver. A V100 GPU is in the same performance class as an Apple M2 Max GPU, which is integrated into a standard MacBook Pro laptop; we note that this is based on published benchmarks and we have not conducted a full end-to-end evaluation on Apple hardware. To make a fair comparison, we use network traces from the WebRTC evaluation to evaluate Reparo.

Metrics. We evaluate peak signal-to-noise ratio (PSNR), structural similarity (SSIM) (Wang et al., 2004), Learned Perceptual Image Patch Similarity (LPIPS) (Zhang et al., 2018), and the percentage of non-rendered frames. PSNR, SSIM, and LPIPS are computed using the PyTorch Image Quality (PIQ) library (Kastryulin et al., 2022) by comparing the decoded frames at the receiver to the original frames at the transmitter. The definition of non-rendered frames differs between baselines and Reparo. For baselines, it is the fraction of frames not displayed due to packet loss on that frame or dependencies on previously undecodable frames. For Reparo, which always attempts to produce a frame, we classify frames with PSNR $< 30\,\mathrm{dB}$ as non-rendered. On our dataset, VP8/9 rarely falls below this threshold unless a frame is undecodable, so this definition is conservative and favors the baselines. Non-rendered frames correlate strongly with QoE, as high rates of such frames lead to freezes and perceptible degradation. All metrics are aggregated over all frames in the validation set. Bitrate is computed by averaging packet sizes (excluding TCP/IP headers) from the recorded network traces over each entire video.

Network Scenarios. We evaluate two scenarios that induce packet loss during transmission: (1) an unreliable network (§4.2) and (2) a rate-limited link (§A.2). In the unreliable-network scenario, packets are randomly dropped due to poor network conditions. We model this using a GE loss channel that alternates between a "good" state with low loss and a "bad" state with high loss, following Tambur's setup (Rudow et al., 2023). The transition probabilities from good$\rightarrow$bad and bad$\rightarrow$good are 0.068 and 0.852, respectively, and the good-state loss rate is 0.04. These parameters, taken from Tambur, approximate aggregate statistics from a large corpus of Microsoft Teams traces (Rudow et al., 2023). We vary the bad-state loss rate to assess performance across loss regimes: 0.25 (low), 0.5 (medium, Tambur's default), and 0.75 (high). The resulting burst-loss statistics are summarized in Tab. 1.

In the rate-limited-link scenario, packet drops occur when the link becomes saturated. We simulate this with a FIFO queue of fixed length (6 KB) and a drain rate of 320 Kbps.

Reparo Parameters. We use a 512×512 frame size and compress it into 32×32 tokens. With a codebook size of 1024, each token requires 10 bits to represent its index. The codebook is trained once across the entire dataset and frozen during evaluation, eliminating the need to transmit it during video conferencing. A frame of tokens is split into 4 packets, with a packet header size of 4 bytes containing a 20-bit frame index, a 2-bit packet index, and a 10-bit packet size. Therefore, to send all the tokens of a frame, each packet

requires 324 bytes, resulting in a default bitrate of 311.04 Kbps (at 30 fps). We use smaller packets to increase the granularity of loss: since the packetizer distributes spatially adjacent tokens across different packets (§3.2.2), a single lost packet removes only a sparse, spatially dispersed subset of tokens, which the loss recovery module can reconstruct more effectively. Larger packets (e.g., 1024–1500 bytes) would cause a single loss event to remove a larger contiguous region of information and degrade reconstruction quality. The packet size can be adjusted for different deployment settings; the current choice of 4 packets per frame balances loss granularity against per-packet header overhead. The tokens in each packet can be dropped up to 50% to match the target bitrate using the "self-drop" mechanism described in §3.2.3. We can further control the bitrate (and visual quality trade-off) by using a different codebook and number of tokens per frame as shown in Fig. 12.

4.2 Performance on Lossy Networks

Visual Quality. We first compare the visual quality of video displayed using baselines and Reparo by evaluating the PSNR, SSIM, and LPIPS under different loss levels. We also vary the target bitrate to evaluate the performance of our method and baseline under different bitrate constraints. We measure the average performance over all frames and the worst 10% frames, representing the overall video conferencing quality and the quality under poor network conditions. All metrics are aggregated across all frames in our evaluation corpus.

As shown in Fig. 5, Reparo achieves better performance with smaller bitrates for all metrics and under all conditions. The poor performance of baselines is caused by freezes of displayed video: during a freeze, the video is stuck at the last rendered frame. In contrast, Reparo maintains a high and stable performance even under high loss levels, thanks to two key design elements. First, Reparo does not have any temporal dependency at the neural codec level. Encoding into and decoding from tokens occur on a frame-by-frame basis without any dependency on a previous frame. Thus, even if a frame's tokens are mostly lost, it could have a lower performance but will not affect subsequent frames whose tokens are received. Second, the loss recovery module uses a deep generative network that leverages domain knowledge of human faces to generate lost tokens. It will only fail to generate accurately if a very large portion of tokens is lost across packets over multiple frames, which is highly unlikely.

We further show the distribution of frame PSNR values across the frames in our evaluation corpus with Reparo and baselines at $\sim$320 Kbps under different packet loss rates in the "bad" state of the GE channel. As shown in Fig. 6, Reparo's distribution and averages of PSNR values are more or less unaffected by the loss level. With Reparo, almost 99% of frames have PSNR values larger than 30 dB. The variance of PSNR values across displayed frames is also much lower with Reparo than the baselines, showing the stability of the quality of the displayed video across loss levels. Specifically, Reparo's frame PSNR values are mostly between 32.5 dB and 37 dB (¿90%)). In contrast, as the loss level becomes higher, the baselines are more likely to experience video freezes, resulting in more frames with low PSNR. These results demonstrate that Reparo is more robust and efficient at recovering from packet losses than current FEC schemes for video conferencing.

Non-Rendered Frames. Another commonly used metric for evaluating FEC approaches is the frequency of non-rendered frames, which can cause freezes in the displayed video. One advantage of Reparo is that it never truly freezes: it always attempts to generate lost tokens and the frame, regardless of the packet loss rate. However, in extreme cases, it may still produce poor generated output. To provide a fair comparison, we define frames with a PSNR of less than 30 dB as "non-rendered frames" for Reparo, since we observed that VP8/9's PSNR rarely drops below 30 dB unless frames are lost and the video stalls. We note that such a definition favors baselines in their comparisons with Reparo since we do not penalize them for low-quality rendered frames.

We evaluated Reparo and baselines at a similar bitrate ($\sim$320 Kbps) under various loss levels. As shown in Fig. 8, Reparo nearly eliminates non-rendered frames under all loss levels, whereas all baselines have a noticeable number of non-rendered frames. This result further demonstrates Reparo's effectiveness in displaying consistently high-quality videos even under severe packet losses, in contrast to current codecs and FEC schemes that cause extended freezes.

4.3 Summary of Other Experiments

In addition to our main evaluation, we conduct several further studies in the appendix to better understand Reparo's behavior under different conditions.

Short Freeze Case Study. As detailed in §A.1 and illustrated in Fig. 9, we examine a short freeze event in which Tambur freezes for eight consecutive frames. Despite experiencing the same packet losses, Reparo continues to render frames and avoids a visible stall. Although one frame temporarily falls below 30 dB due to heavy loss, Reparo quickly recovers once subsequent tokens arrive, maintaining motion continuity and overall visual quality.

Performance on Rate-Limited Links. §A.2 evaluates both systems over a fixed-capacity bottleneck link. Because Reparo produces identically sized frames, its bitrate closely matches the target rate. In contrast, VP9+Tambur generates large keyframes and variable-size P-frames that can overflow the bottleneck queue, leading to packet drops. As shown in

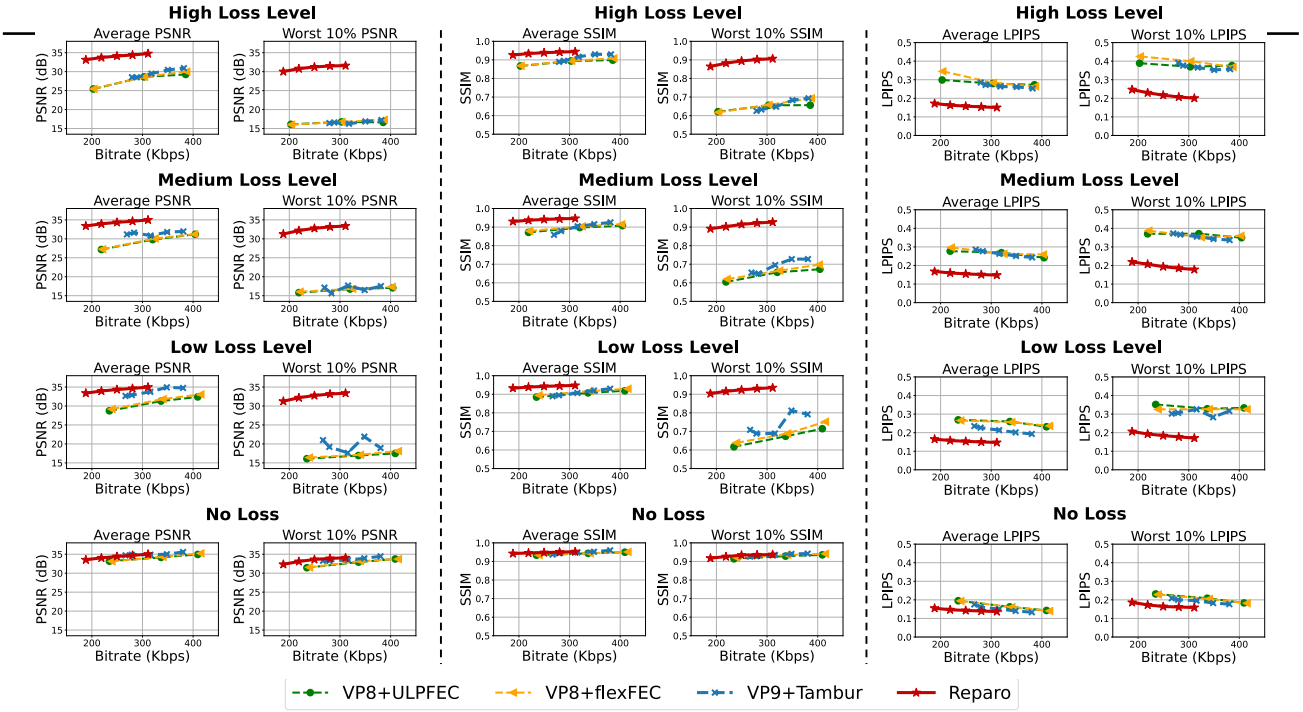

Figure 5: We report the average and worst 10% PSNR, SSIM and LPIPS of baselines and Reparo under different loss levels. PSNR and SSIM are the higher the better, and LPIPS is the lower the better. We vary the target bitrate of Reparo and baselines to cover different achieved bitrates. Reparo's visual quality is significantly better than the baselines under all lossy conditions while achieving similar performance when there is no loss.

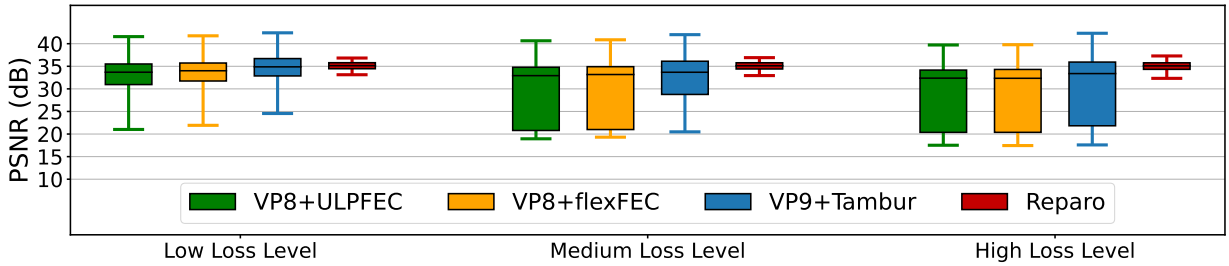

Figure 6: PSNR distribution across frames with baselines and Reparo under different packet loss rates (for a bitrate of ∼320 Kbps). The box denotes the 25th and 75th percentile PSNR, the line inside the box denotes the median PSNR while the whiskers denote average PSNR ± 1.5×standard deviation. Reparo maintains its PSNR within a narrow band around 35 dB regardless of the loss level while Tambur's worst frames drop to less than 20 dB PSNR at higher loss rates.

Fig. 10 and Fig. 11, Reparo 's video quality increases steadily with higher target bitrates, whereas VP9+Tambur's quality degrades once its keyframes no longer fit into the link buffer.

Adaptability Through Codec Variants. §A.3 further explores Reparo 's adaptability by varying the number of tokens, the codebook size, and the number of jointly encoded frames. These options allow Reparo to operate across a wide range of bitrates and latency settings. As shown in Fig. 12, increasing the number of tokens or the codebook size improves quality at higher bitrates, while jointly encoding multiple frames reduces bitrate at the cost of modest added delay.

Latency Breakdown. §A.3 reports a detailed latency analysis summarized in Tab. 2. Combined with typical network delays, Reparo 's end-to-end latency remains under 100 ms, staying within recommended limits for interactive video applications.

Qualitative Results. We compare Reparo with Tambur qualitatively. Please refer to this `link` for results.

5 Limitations

Although Reparo offers several key advantages over past work, it also has some limitations. First, the current implementation of Reparo is in PyTorch, and uses transformers which are computationally more intensive than traditional video codecs and FEC-based methods (Dosovitskiy et al., 2021; Arnab et al., 2021). It requires GPUs in the same performance class as an Apple M2 Max GPU to operate in real time. This limits the range of devices on which Reparo can be deployed, and the current implementation is not suitable for low-end devices such as smartphones or tablets. However, machine learning models can be sped up for edge devices using more efficient model architectures (Howard

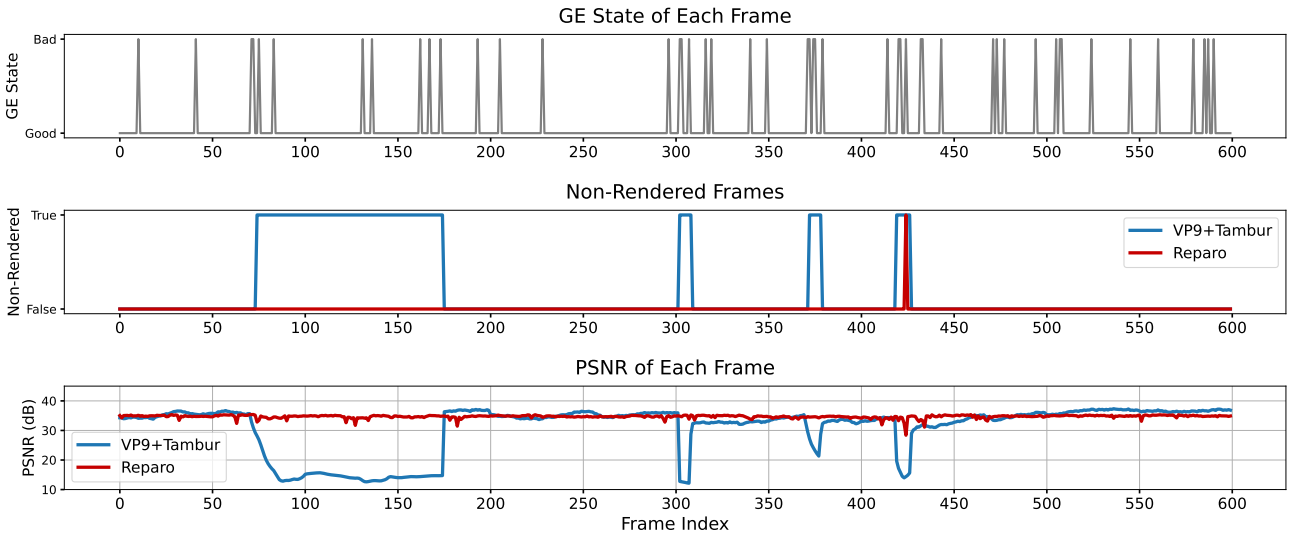

Figure 7: Timeseries comparing Tambur and Reparo on one video and loss pattern. Tambur experiences short freezes every time a set of frames is lost with a corresponding decrease in PSNR. Reparo continues rendering frames and its visual quality is a lot more stable throughout the interval.

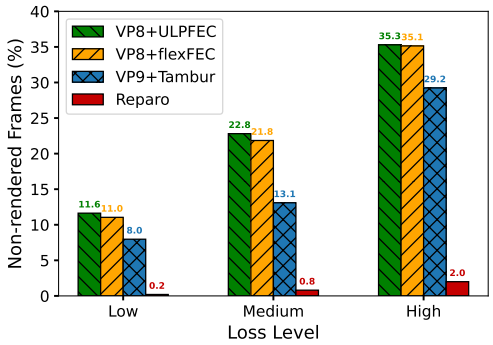

Figure 8: Comparison of percentage of non-rendered frames between Reparo and baselines. All baselines experience many more non-rendered frames than Reparo at all loss levels.

et al., 2017; Tan & Le, 2019; Elsken et al., 2019; Rao et al., 2021), hardware design (Gale et al., 2020; Han et al., 2017; Zhang et al., 2020), and techniques such as knowledge distillation (Hinton et al., 2015). We leave an investigation of such optimizations to future work. We also note that over time more powerful GPUs are integrated in edge devices naturally paving the way for running complex neural networks on them. Second, Reparo learns a dictionary of tokens specific to the domain of interest, which is video conferencing for the purpose of this paper. While Reparo can potentially be extended beyond video conferencing to other domains, this will require learning a dictionary for each new domain. As such it is more specialized and less general than traditional FEC codecs. Third, our current prototype operates at 512×512. A key property of generative neural codecs is that the compressed token count can be partially decoupled from the pixel resolution: for higher-resolution scenarios (e.g., 1080p), a codec can be trained with a larger spatial downsampling ratio, yielding fewer tokens per pixel while maintaining strong perceptual reconstruction quality. This would keep the token budget and recovery computation cost closer to the 512×512 setting. Finally, our PSNR-based non-rendered frame threshold is a conservative proxy that may not capture perceptual artifacts like temporal flickering or uncanny-valley effects. We leave exploration of higher-resolution codecs and richer perceptual and temporal consistency metrics to future work. We provide additional discussion and limitations in App. C. Despite these limitations, Reparo represents a promising approach to loss-resilient video conferencing. Future research may focus on addressing these limitations and making Reparo more accessible to a wider range of devices and different video-based applications.

6 Conclusion

We present Reparo, a novel loss-resilient generative video conferencing architecture that uses generative deep learning models to reconstruct missing information without sending redundant packets or relying on retransmissions. Instead, the receiver reconstructs missing information using its knowledge of how visual objects look and relate to each other. Our approach offers several advantages, including maintaining a constant bit rate, easy adaptation to any target bitrate, and one-way communication between the transmitter and receiver. We evaluate Reparo on a large and diverse corpus of publicly available video conferencing videos and show that it consistently outperforms multiple FEC baselines including Tambur, a state-of-the-art loss-resilient video conferencing platform based on streaming FEC. Reparo significantly improves over them under different loss levels, while also mostly eliminating video freezes. Our approach presents a promising solution to the challenges of real-time video conferencing applications, and we believe it opens up exciting possibilities for further research in this area.

REFERENCES

AV1 bitstream & decoding process specification. http://aomedia.org/av1/specification/.

Ringmaster. https://github.com/microsoft/ringmaster.

WebRTC. https://webrtc.googlesource.com/src.

Apple Inc. Apple M2 Max — technical specifications, 2023. URL https://support.apple.com/en-us/111901. Accessed: 2025.

Arnab, A., Dehghani, M., Heigold, G., Sun, C., Lučić, M., and Schmid, C. Vivit: A video vision transformer. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 6836–6846, 2021.

Bankoski, J., Wilkins, P., and Xu, Y. Technical overview of VP8, an open source video codec for the web. In *2011 IEEE International Conference on Multimedia and Expo*, pp. 1–6. IEEE, 2011.

Begen, A. Rtp payload format for 1-d interleaved parity forward error correction (fec). Technical report, 2010.

Boyce, J. M. Packet loss resilient transmission of mpeg video over the internet. *Signal Processing: Image Communication*, 15(1-2):7–24, 1999.

Carlucci, G., De Cicco, L., Holmer, S., and Mascolo, S. Analysis and design of the google congestion control for web real-time communication (webrtc). In *Proceedings of the 7th International Conference on Multimedia Systems*, pp. 1–12, 2016.

Chang, H., Zhang, H., Jiang, L., Liu, C., and Freeman, W. T. Maskgit: Masked generative image transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11315–11325, 2022.

Chang, H., Zhang, H., Barber, J., Maschinot, A., Lezama, J., Jiang, L., Yang, M.-H., Murphy, K., Freeman, W. T., Rubinstein, M., et al. Muse: Text-to-image generation via masked generative transformers. *arXiv preprint arXiv:2301.00704*, 2023.

Cheng, Y., Zhang, Z., Li, H., Arapin, A., Zhang, Y., Zhang, Q., Liu, Y., Zhang, X., Yan, F. Y., Mazumdar, A., Feamster, N., and Jiang, J. Grace: Loss-resilient real-time video through neural codecs, 2023.

Dasari, M., Kahatapitiya, K., Das, S. R., Balasubramanian, A., and Samaras, D. Swift: Adaptive video streaming with layered neural codecs. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pp. 103–118, Renton, WA, April 2022. USENIX Association. ISBN 978-1-939133-27-4. URL https://www.usenix.org/conference/nsdi22/presentation/dasari.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *Int. Conf. on Learning Representations (ICLR)*, 2021.

Elsken, T., Metzen, J. H., and Hutter, F. Neural architecture search: A survey. *The Journal of Machine Learning Research*, 20(1):1997–2017, 2019.

Esser, P., Rombach, R., and Ommer, B. Taming transformers for high-resolution image synthesis. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 12873–12883, 2021.

Gale, T., Zaharia, M., Young, C., and Elsen, E. Sparse gpu kernels for deep learning. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–14. IEEE, 2020.

Han, S., Kang, J., Mao, H., Hu, Y., Li, X., Li, Y., Xie, D., Luo, H., Yao, S., Wang, Y., et al. Ese: Efficient speech recognition engine with sparse lstm on fpga. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 75–84, 2017.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

He, K., Chen, X., Xie, S., Li, Y., Dollár, P., and Girshick, R. Masked autoencoders are scalable vision learners. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 16000–16009, June 2022.

Hinton, G., Vinyals, O., and Dean, J. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

Isola, P., Zhu, J.-Y., Zhou, T., and Efros, A. A. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1125–1134, 2017.

Karimi, P. *Bridging the Gap Between Real-time Video and Backlogged Traffic Congestion Control*. PhD thesis, Massachusetts Institute of Technology, 2023. URL https://www.proquest.com/docview/3030939628.

Karimi, P., Fouladi, S., Sivaraman, V., and Alizadeh, M. Vidaptive: Efficient and responsive rate control for real-time video on variable networks, 2024. URL https://arxiv.org/abs/2309.16869.

Karimi, P., Fouladi, S., Sivaraman, V., and Alizadeh, M. Tight Loops, Smooth Streams: Responsive Congestion Control for Real-Time Video. In Argyraki, K. and Panda, A. (eds.), *1st New Ideas in Networked Systems (NINeS 2026)*, volume 139 of *Open Access Series in Informatics (OASIcs)*, pp. 9:1–9:29, Dagstuhl, Germany, 2026. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN 978-3-95977-414-7. doi: 10.4230/OASIcs.NINeS.2026.9. URL https://drops.dagstuhl.de/entities/document/10.4230/OASIcs.NINeS.2026.9.

Karras, T., Aila, T., Laine, S., and Lehtinen, J. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*, 2017.

Karras, T., Laine, S., and Aila, T. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 4401–4410, 2019.

Kastryulin, S., Zakirov, J., Prokopenko, D., and Dylov, D. V. Pytorch image quality: Metrics for image quality assessment, 2022. URL https://arxiv.org/abs/2208.14818.

Khani, M., Sivaraman, V., and Alizadeh, M. Efficient video compression via content-adaptive super-resolution. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 4521–4530, 2021.

Kuznetsova, A., Rom, H., Alldrin, N., Uijlings, J., Krasin, I., Pont-Tuset, J., Kamali, S., Popov, S., Malloci, M., Kolesnikov, A., et al. The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale. *International Journal of Computer Vision*, 128(7):1956–1981, 2020.

Lee, D., Kim, C., Kim, S., Cho, M., and Han, W. Autoregressive image generation using residual quantization. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

Li, T., Chang, H., Mishra, S. K., Zhang, H., Katabi, D., and Krishnan, D. Mage: Masked generative encoder to unify representation learning and image synthesis. *arXiv preprint arXiv:2211.09117*, 2022.

Li, T., Katabi, D., and He, K. Self-conditioned image generation via generating representations. *arXiv preprint arXiv:2312.03701*, 2023a.

Li, Z., Hu, Y., Guo, H., Zhang, Q., Song, C., Ji, R., Jiang, B., Zhang, Z., Hu, M., Wang, L., et al. Context-diverse neural video codec. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023b.

Liu, H., Chen, Z., Yuan, Y., Mei, X., Liu, X., Mandic, D., Wang, W., and Plumbley, M. D. Audioldm: Text-to-audio generation with latent diffusion models. *arXiv preprint arXiv:2301.12503*, 2023.

Loshchilov, I. and Hutter, F. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Lu, G., Ouyang, W., Xu, D., Zhang, X., Cai, C., and Gao, Z. DVC: An end-to-end deep video compression framework. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 11006–11015, 2019.

MacKay, D. J. Fountain codes. *IEE Proceedings-Communications*, 152(6):1062–1068, 2005.

Mentzer, F., Agustsson, E., Ballé, J., Minnen, D., Johnston, N., and Toderici, G. Neural video compression using gans for detail synthesis and propagation. In *Proceedings of the European Conference on Computer Vision*, 2022.

Mukherjee, D., Han, J., Bankoski, J., Bultje, R., Grange, A., Koleszar, J., Wilkins, P., and Xu, Y. A technical overview of VP9, the latest open-source video codec. *SMPTE Motion Imaging Journal*, 124(1):44–54, 2015.

Netravali, R., Sivaraman, A., Das, S., Goyal, A., Winstein, K., Mickens, J., and Balakrishnan, H. Mahimahi: accurate {Record-and-Replay} for {HTTP}. In *2015 USENIX Annual Technical Conference (USENIX ATC 15)*, pp. 417–429, 2015.

NVIDIA Corporation. NVIDIA V100 Tensor Core GPU datasheet, 2020. URL https://images.nvidia.com/content/technologies/volta/pdf/volta-v100-datasheet-update-us-1165301-pdf. Accessed: 2025.

Rao, Y., Zhao, W., Liu, B., Lu, J., Zhou, J., and Hsieh, C.-J. Dynamicvit: Efficient vision transformers with dynamic token sparsification. *Advances in neural information processing systems*, 34:13937–13949, 2021.

Razavi, A., Van den Oord, A., and Vinyals, O. Generating diverse high-fidelity images with vq-vae-2. *Advances in neural information processing systems*, 32, 2019.

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

Rombach, R., Blattmann, A., Lorenz, D., Esser, P., and Ommer, B. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10684–10695, 2022.

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

Santurkar, S., Schmidt, L., Tsipras, D., and Madry, A. Generative compression. *arXiv preprint arXiv:1703.01467*, 2017.

Schwarz, H., Marpe, D., and Wiegand, T. Overview of the scalable video coding extension of the h. 264/avc standard. *IEEE Transactions on circuits and systems for video technology*, 17(9):1103–1120, 2007.

Sivaraman, V., Karimi, P., Venkatapathy, V., Khani, M., Fouladi, S., Alizadeh, M., Durand, F., and Sze, V. Gemino: Practical and robust neural compression for video conferencing. *arXiv preprint arXiv:2209.10507*, 2022.

Sullivan, G. J., Ohm, J.-R., Han, W.-J., and Wiegand, T. Overview of the high efficiency video coding (HEVC) standard. *IEEE Transactions on circuits and systems for video technology*, 22(12):1649–1668, 2012.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826, 2016.

Tan, M. and Le, Q. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pp. 6105–6114. PMLR, 2019.

Union, I. T. ITU-T G.1010: End-user multimedia QoS categories. In *Series G: Transmission Systems and Media, Digital Systems and Networks*, 2001.

van den Oord, A., Vinyals, O., and Kavukcuoglu, K. Neural discrete representation learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Wan, R., Zheng, Q., and Fan, Y. M3-cvc: Controllable video compression with multimodal generative models. *arXiv preprint arXiv:2411.15798*, 2024.

Wang, T.-C., Mallya, A., and Liu, M.-Y. One-shot free-view neural talking-head synthesis for video conferencing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10039–10049, 2021a.

Wang, T.-C., Mallya, A., and Liu, M.-Y. One-shot free-view neural talking-head synthesis for video conferencing. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10039–10049, 2021b.

Wang, Z., Bovik, A. C., Sheikh, H. R., and Simoncelli, E. P. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

Xia, Z., Li, H., and Jiang, J. Loss-tolerant neural video codec aware congestion control for real-time video communication. *arXiv preprint arXiv:2411.06742*, 2025.

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

Yi, F., Xu, J., Shao, J., Zhang, C., and Li, X. Conditional video generation for high-efficiency video compression, 2025. URL https://arxiv.org/abs/2507.15269.

Yu, J., Li, X., Koh, J. Y., Zhang, H., Pang, R., Qin, J., Ku, A., Xu, Y., Baldridge, J., and Wu, Y. Vector-quantized image modeling with improved vqgan. *arXiv preprint arXiv:2110.04627*, 2021.

Zhang, A., Hu, Y., Wang, C., et al. Nier: Practical neural-enhanced low-bitrate video conferencing. In *Proceedings of the ACM SIGCOMM Conference*, 2025.

Zhang, R., Isola, P., Efros, A. A., Shechtman, E., and Wang, O. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 586–595, 2018.

Zhang, Z., Wang, H., Han, S., and Dally, W. J. Sparch: Efficient architecture for sparse matrix multiplication. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 261–274. IEEE, 2020.

Zheng, H. and Boyce, J. An improved udp protocol for video transmission over internet-to-wireless networks. *IEEE Transactions on Multimedia*, 3(3):356–365, 2001.

Table 1: Statistics of bursty lossy networks emulated using GE loss channel.

	Low Loss Level	Medium Loss Level	High Loss Level
Loss rate in bad state	25%	50%	75%
Time in bad state	7.4%	7.4%	7.4%
Avg. loss rate	5.6%	7.4%	9.3%

A Deep Dive

A.1 A Detailed Example

To better understand Reparo's benefits come from, we present a time series of loss patterns, non-rendered frames, and PSNR values for Reparo and Tambur over a 30-second window in Fig. 7 for a particular video sequence. The loss level is set to medium (packet loss probability of 0.5 in the bad state). The sequence of lost frames starting at frame index 71 causes Tambur to experience an extended freeze between frame 72 and frame 177, even though many frames in that timeframe were not lost. This is due to temporal dependencies between video frames, where frames are compressed based on the differences between them. As a result, a lost frame can lead to subsequent undecodable frames (even when they're received successfully) until the encoder and decoder are reset using a keyframe. As expected, Tambur exhibits much lower PSNR ($\sim$15 dB) during that timeframe between frames 72 and 177. Tambur then forces the encoder to transmit a keyframe to resume the video stream. Subsequent frames' PSNR values go back to what they were prior to the freeze period. If such a keyframe is also lost (which is more likely because a keyframe is much larger than normal frames and contains more packets since it is compressed independently of its adjacent frames), it could cause long freezes that span several seconds.

In contrast, Reparo is much more stable in PSNR and rarely experiences non-rendered frames, even during periods of loss. Reparo may generate one or more frames with low PSNR if it loses many tokens, as happens at frame 424. However, its per-frame decoding structure ensures that its visual quality quickly recovers as tokens for future frames start coming in.

To gain a more comprehensive understanding of the effects of packet loss events, we examine a short freeze event of Tambur spanning 8 frames in greater detail and compare it to Reparo in Fig. 9. This figure shows lost frames, non-rendered frames, frame PSNR values as well as visuals of the displayed frames in that time interval. As depicted in the figure, part of the 3rd, 4th, and 5th frames are initially lost, followed by the loss of the 8th frame. Tambur does not render any frames between the 3rd and 10th frames, as is evident from the "non-rendered" frames line and the unchanged video frames in the visual strip beneath. Additionally, the forced keyframe (frame 11) and subsequent frame 12 have slightly lower PSNR due to the larger size of the keyframe, which typically has a lower quality to meet the target bitrate when compressed without any temporal dependency. In contrast, Reparo does not experience such a prolonged freeze, as evidenced by the "non-rendered frames" row and the visual strip. Although Reparo produces a lower PSNR frame at the 8th frame, it rapidly recovers once later frames receive sufficient packets and tokens for high-quality generation.

A.2 Performance on Rate-Limited Networks

In this section, we consider the packet loss caused by a *rate-limited bottleneck link* when it saturates. One advantage of Reparo is its ability to match and transmit at different target bitrates easily by simply varying the self-drop rate. This is because, unlike traditional temporal-dependent codecs, Reparo does not need to transmit keyframes periodically. Instead, every frame is encoded into a set of tokens with the same size across frames. As shown in Fig. 10, VP9+Tambur needs to transmit a keyframe periodically, causing spikes in its per-frame sizes. Even the P-frames in VP9 show quite a bit of variance in their sizes. In contrast, Reparo can always maintain a constant size across frames and consequently, constant bitrate because its neural codec encodes each frame with the same number of tokens.

Such a stable bitrate improves Reparo's performance over fixed-capacity bottleneck links. To simulate such a link, we use a FIFO queue with a constant (drain) rate of r Kbps. The size of the queue is set to $0.15 \times r$, as such a queue will introduce a 150 ms delay, which is the upper bound of industry recommendations for interactive video conferencing (Union, 2001). Packets are queued first and drained at the desired link rate. When the FIFO queue becomes full, subsequent packets will be dropped. In Fig. 11, we set r to 320 Kbps and show the average PSNR of Reparo and VP9+Tambur with different target bitrates for each codec. Note that the target bitrate for VP9+Tambur typically does not match the actual bitrate: it is the input parameter for the VP9 codec to encode a video. As a result, the actual bitrate of VP9+Tambur can be much larger than the target bitrate of VP9 depending on the encoding speed and quality parameters. Also, Tambur's parity packets typically introduce 50% to 60% bandwidth overhead, further inflating the actual bitrate of VP9+Tambur. For example, the 75 Kbps target bitrate corresponds to an actual average bitrate of 211 Kbps. As a result, we only vary the target bitrate supplied to VP9+Tambur up to 200 Kbps because beyond that its actual bitrate with FEC overheads overshoots the link rate and causes a lot of packet drops. On the other hand, Reparo's actual bitrate can exactly match the target bitrate.

As shown in Fig. 11, the average PSNR achieved by Reparo increases as the target bitrate is increased. This is expected because fewer tokens are "self-dropped", allowing for better reconstruction. However, while the PSNR of Tambur initially increases as the target bitrate is increased, it begins to decrease when the target bitrate is set to 120 Kbps. This occurs because even with a small target bitrate, the size

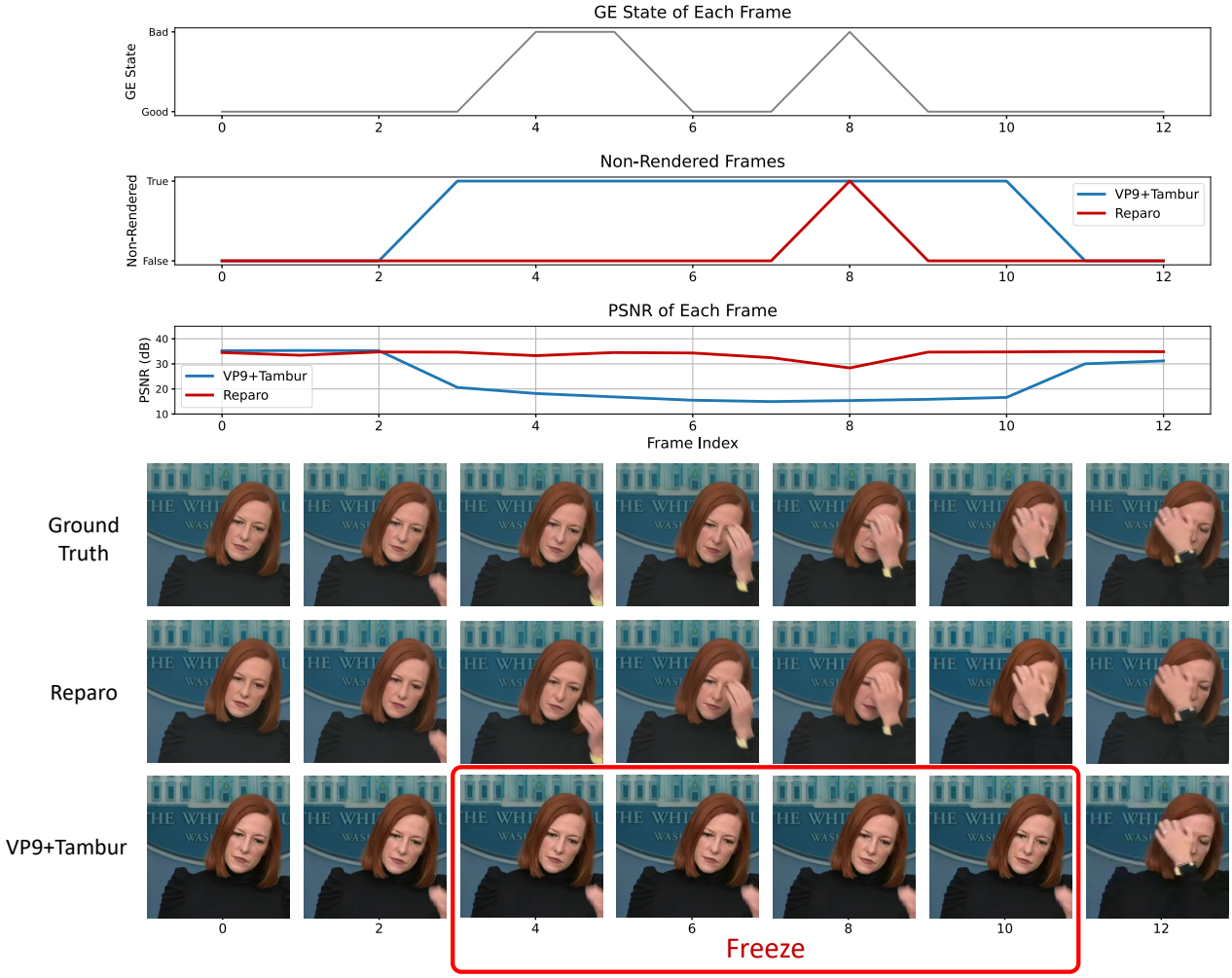

Figure 9: Qualitative results of Tambur and Reparo during a Tambur's short freeze of 8 frames. The GE loss channel is in a "bad" state at frames 4, 5, 6, and 8, causing packet losses for both VP9+Tambur and Reparo. Tambur completely freezes from frames 3 to 10 because of lost packets, leading to very low PSNR. On the other hand, though Reparo experiences the same GE loss state as Tambur, it generates most of the frames and maintains a high PSNR. Even for the frame under 30 PSNR, it still produces reasonable output and tracks the hand movement accurately.

of a keyframe across all its packets with VP9 can be much larger than the total number of bytes that the queue can hold. Consequently, many packets of this keyframe may be lost. Additionally, when a keyframe is lost, Tambur will force another keyframe, causing the queue to remain full and preventing any frames from being transmitted, resulting in a frozen video over long durations. As the target bitrate is increased further and this issue with keyframes becomes more pronounced, VP9+Tambur's average PSNR worsens.

In practice, congestion control protocols like GCC (Carlucci et al., 2016) are used to adapt the encoder's target bitrate based on network observations such as latency and loss. However, this experiment shows that choosing the appropriate target bitrate for Tambur is much more challenging than for Reparo. For VP9+Tambur, the adaptation protocol must be conservative and operate in a lower bitrate range to limit packet drops. In contrast, Reparo can continue to benefit from larger target bitrates as long as they are smaller than the link capacity, and the best performance is achieved by setting the target bitrate near the link capacity.

A.3 Other Results

Reparo Ablation Study. To allow Reparo to operate in different bitrate regimes, we can adjust its hyper-parameters. For example, we can compress t adjacent frames into the same fixed size $h \times w$ tokens, which reduces the effective bitrate by a factor of t at the cost of an additional latency of $t-1$ frames. We can also modify the number of residual blocks used in the encoder and decoder, which changes the number of tokens to represent a frame. More tokens per frame correspond to better PSNR and higher bitrate due to better representational power. We can also use different codebook sizes; larger codebook sizes produce higher PSNR at the

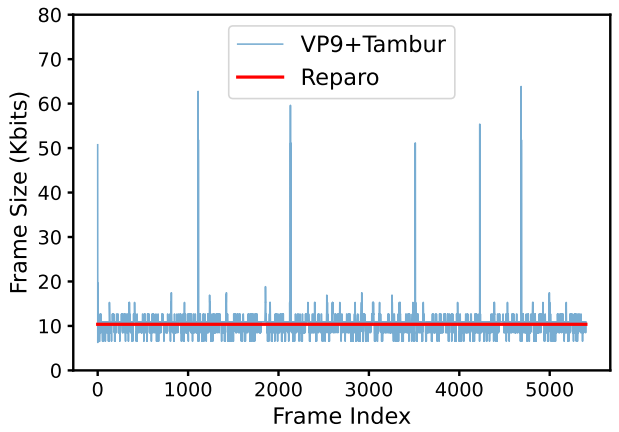

Figure 10: Per frame sizes of VP9+Tambur and Reparo for a 3 minute video. Reparo maintains the same frame size across all frames while VP9 shows variance both across adjacent frames and across periodic keyframes that are large.

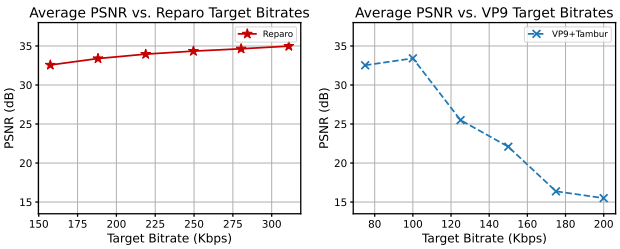

Figure 11: Average PSNR of Reparo and Tambur with different target bitrates for a fixed link capacity of 320 Kbps. Reparo's average PSNR improves as the target bitrate is increased. However, VP9+Tambur starts experiencing loss in its fixed-size queue beyond a target bitrate of $120\,\mathrm{Kbps}$ **due to large keyframes that do not fit in the queue.**

cost of a larger bitrate. In Fig. 12, we show the PSNR-bitrate curve of Reparo under different hyper-parameters with a low loss level, demonstrating that Reparo can be adapted to a large range of bitrates by varying the codec and loss recovery module trained under different hyper-parameters. For example, Reparo can choose to encode two frames together at the cost of $33\,\mathrm{ms}$ higher latency and achieve almost half bitrate (red curve and orange curve). Reparo can also use a larger codebook to achieve higher PSNR at the cost of more bits needed to encode each token index (red curve and blue curve). By default, we use the middle red curve (t=1, number of tokens per frame=32×32, codebook size=1024) for 30 fps 512×512 videos in our main experiments.

Table 2: Latency breakdown for different parts of Reparo. The encoder and packetization are at the transmitter side, while the loss recovery and decoder are at the receiver side.

	Encoder	Packetization	Loss Recovery	Decoder
Latency (ms)	14.1±0.1	0.5±0.009	17.8±1.0	13.1±0.3

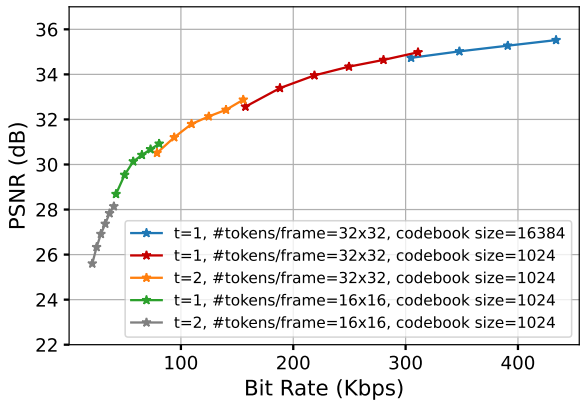

Figure 12: Variants of Reparo that operate in different bitrate regimes. Reparo achieves different bitrates by varying the number of tokens per frame, its codebook size, and the number of frames jointly encoded.

Latency. In Tab. 2, we present the latency of different modules in Reparo. The neural codec and loss recovery modules of Reparo have higher encoding and decoding latencies compared to traditional FEC schemes, since they require heavy computation. For our implementation, the total inference delay incurred by Reparo is $45.5\,\mathrm{ms}$. With typical network queuing delays of $50\,\mathrm{ms}$, the end-to-end delay of Reparo is less than $100\,\mathrm{ms}$, which meets the industry recommendation of $150\,\mathrm{ms}$ for maximum tolerable latency for interactive video applications (Union, 2001).

B Implementation Details

Because of GPU memory limitation, we adopt a two-stage training recipe for Reparo similar to many prior approaches (Li et al., 2022; Esser et al., 2021). We first train our VQGAN codec which encodes each video frame into discrete tokens *without* any losses. We then *fix* the VQGAN codec and train the loss recovery module on the discrete tokens with self-dropping and packet loss. In this section, we describe the neural network structure and training schemes of Reparo's neural codec and loss recovery module, as well as the design of the bitrate controller in detail.

B.1 Neural Codec

Model Structure. We use a CNN-based VQGAN (Esser et al., 2021) encoder and quantizer to tokenize the $3\times512\times512$ input frame to $128\times32\times32$ quantized features, where 128 is the number of channels of the quantized features. It then uses a codebook to quantize the features by finding the nearest neighbor of each feature in the codebook. The codebook is a 1024×128 matrix by default, with 1024 entries, each of which uses a 128-dimensional feature. The decoder operates on the quantized features and reconstructs the $3\times512\times512$ video frame. The encoder

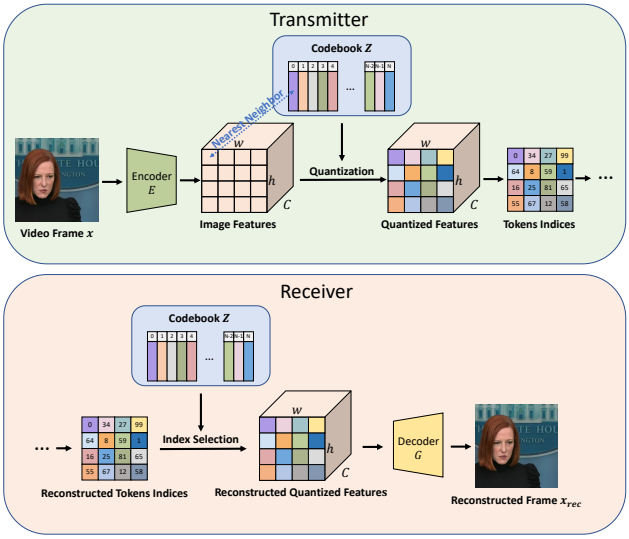

Figure 13: Token-based neural codec. The transmitter first uses the encoder to convert patches from video frames into features. It then uses a codebook to quantize the features into tokens by finding the nearest neighbor of each feature in the codebook. The receiver first maps the received and generated tokens to back-to-image features using the codebook. It then uses a decoder to reconstruct the video frame.

consists of 5 blocks and each block consists of 2 residual blocks which follow standard ResNet's residual block design (He et al., 2016). After each block in the encoder, the feature vector is down-sampled by 2 using average pooling. The quantizer then maps each pixel of the encoder's output feature map to the nearest token (based on L_2 distance) in the codebook Z with $N=1024$ entries, each entry with 128 channels. The decoder consists of another 5 blocks where each encoder block has 2 residual blocks. After each block in the decoder, the feature map is up-sampled by 2 using bicubic interpolation. The tokenizer consists of 23.8M parameters and the detokenizer consists of 30.5M parameters.

Training schemes. We follow the original VQGAN training recipe (Esser et al., 2021) to train the VQGAN. We use a vector-quantize loss between the image features and quantized tokens that nudges the image features towards the tokens that they map to, a reconstruction loss (L_1) between the input and final reconstructed frame, a perceptual loss (Zhang et al., 2018) between the input and reconstructed frame, and a discriminative loss (Isola et al., 2017) between the input and reconstructed frame. Detailed descriptions of the losses can be found in the VQGAN paper (Esser et al., 2021).

We use the officially released VQGAN encoder and decoder pre-trained on OpenImages (Kuznetsova et al., 2020) to initialize our codec whenever possible. OpenImages is a large-scale image dataset consisting of ∼9M natural images. We observe that such initialization largely speeds up our training (takes ∼ 10 epochs to converge), but we also note that training from scratch on our pre-training face datasets can achieve similar performance with a much longer training time (∼200 epochs). We train our neural codec using a constant learning rate and train it until there is no substantial change in the training loss. Please refer to Tab. 3 for the training recipe of our VQGAN codec.

Table 3: VQGAN codec training setting.

Parameter	Value
Optimizer	Adam (Loshchilov & Hutter, 2017)
Base Learning Rate	1e-4
Weight Decay	0
Optimizer Momentum	$\beta_1,\beta_2=0.5,0.9$
Batch Size	24
Learning Rate Schedule	Constant (Loshchilov & Hutter, 2016)
Warmup Epochs	0
Gradient Clip	0
Dropout	0

B.2 Loss Recovery Module

Model Structure. The major component of our loss recovery module is a spatio-temporal ViT network. In our default setting, the input tokens are of shape $C\times T\times h\times w$, where $C=768$, $T=6$, $h=32$, $w=32$. We use two separate learnable position embeddings, one for time and one for space, which we add together to provide each input token its positional information. We then adopt a standard spatio-temporal ViT architecture (Dosovitskiy et al., 2021), which consists of a stack of spatio-temporal Transformer blocks (Vaswani et al., 2017). Each spatio-temporal block consists of a spatial block and a temporal block. Each of the two blocks independently consists of a multi-head self-attention block and a multi-layer perceptron (MLP) block. In total, we use 20 spatio-temporal Transformer blocks. The number of heads in each multi-head self-attention layer is 12, and the MLP ratio is 4. The embedding dimension throughout the Transformer is 768. Our spatio-temporal ViT consists of 172M parameters. We note that more Transformer blocks, more heads in the self-attention layer, and a larger embedding dimension can further improve the performance of Reparo , but they also introduce more computation overheads.

Training schemes. Tab. 4 provides the training recipe for our spatio-temporal ViT for loss recovery. The self-drop rate is sampled from a truncated Gaussian distribution from 0 to 0.6 and centered at 0.3, with a standard deviation of 0.3. The packet loss rate is uniformly sampled from 0 to 0.8.

B.3 Bitrate Controller

Reparo employs self-dropping to drop a fixed fraction of tokens across all packets of a frame to achieve the target bitrate. For example, if the target bitrate is 200 Kbps and the

Table 4: Loss recovery module training setting. The model is trained with Label Smoothing (Szegedy et al., 2016), Adam (Loshchilov & Hutter, 2017), and a Cosine Decay schedule (Loshchilov & Hutter, 2016).

Parameter	Value
Optimizer	Adam
Learning Rate	1.5e-5
Weight Decay	0.05
Optimizer Momentum	$\beta_1,\beta_2=0.9,0.95$
Batch Size	24
Learning Rate Schedule	Cosine Decay
Warmup Epochs	10
Training Epochs	200
Gradient Clip	3.0
Label Smoothing	0.1
Dropout	0.1
Min. Self-Drop Rate	0
Max. Self-Drop Rate	0.6
Self-Drop Rate Mode	0.3
Self-Drop Rate Std. Dev.	0.3
Min. Packet Loss Rate	0
Max. Packet Loss Rate	0.8

bitrate when transmitting all tokens is $300\ \mathrm{Kbps}$, the bitrate controller will sample one-third of the tokens in each packet to drop.

To minimize the impact of self-dropping on the loss recovery module, we drop tokens randomly in each packet, so that the dropped tokens are distributed uniformly in space and time. However, randomly dropping tokens in each packet requires telling the receiver which tokens are dropped, leading to bandwidth overheads. Otherwise, the receiver will be confused about the position of each received token in the $h \times w$ token map.

To address this issue, we deterministically sample the tokens to be self-dropped in each packet based on the frame index and packet index. We achieve this by setting the random seed for pseudo-random self-dropping sampling in a packet to $4 \times$ frame index $+$ packet index. Consequently, the receiver compares the received packet size to the expected packet size to identify how many tokens were lost. The receiver then decodes the locations of the lost tokens by simulating the self-drop procedure on its end by repeating the pseudo-random sampling procedure with the same seed and drop rate as the transmitter.

At the start of a video conference, the transmitter selects the variant of the codec and loss recovery module to use based on the target bitrate and synchronizes this information with the receiver. It also communicates the expected number of tokens per packet and frame during this process. Once the variant is established, it can adapt to bitrate changes of up to 50% with self-dropping. If the target bitrate changes significantly, the transmitter selects a new variant and notifies the receiver.

C Additional Discussions

Higher Bandwidth Regimes. Our evaluation focuses on 200–400 Kbps, where conferencing is most loss-sensitive and FEC tuning is hardest. At higher bitrates, Reparo can reduce the self-drop rate or use higher-quality codec variants—larger codebook or more tokens per frame (see Fig. 12)—to improve fidelity while maintaining loss resilience.

Codebook Distribution. The codebook and model weights ship with the conferencing application, similar to how VP8/VP9 codec libraries are bundled. No per-session codebook transmission is needed; session setup only negotiates which codec variant to use (codebook size, token count, resolution), analogous to standard codec capability negotiation.

Multi-Stream Deployment. Our system targets the single-person webcam scenario. Multi-stream group meetings would require sharing the GPU across streams via accelerator scheduling, lighter per-stream recovery models, or server-side offloading. We leave multi-stream deployment to future work.

Integration with Enhancement Pipelines. Reparo is orthogonal to video enhancement techniques such as background removal and noise reduction. Since it produces standard RGB frames, enhancements can be applied either before encoding at the sender or after decoding at the receiver.