Reparo: Loss-Resilient Generative Codec for Video Conferencing

Tianhong Li Vibhaalakshmi Sivaraman Lijie Fan Mohammad Alizadeh Dina Katabi

MIT CSAIL

Abstract

Loss of packets in video conferencing often results in poor quality and video freezing. Attempting to retransmit the lost packets is usually not practical due to the requirement for real-time playback. Using Forward Error Correction (FEC) to recover the lost packets is challenging since it is difficult to determine the appropriate level of redundancy. In this paper, we propose a framework called Reparo for creating loss-resilient video conferencing using generative deep learning models. Our approach involves generating missing information when a frame or part of a frame is lost. This generation is conditioned on the data received so far, and the model's knowledge of how people look, dress, and interact in the visual world. Our experiments on publicly available video conferencing datasets show that Reparo outperforms state-of-the-art FEC-based video conferencing in terms of both video quality (measured by PSNR) and video freezes.

1 Introduction

Traditional video codecs create a strong dependency between encoded frames. For exampe, P-frames depend on previous frames, and to decode them, the receiver must first decode the frames on which it depends. If those frames cannot be decoded due to lost packets, the receiver stalls and cannot proceed, causing the video to freeze. The only way to make progress is to either receive the missing data and decode the needed frames, or to give up and request a new key frame (called I-frame). The I-frame effectively resets the state of the decoder, and subsequent frames can then be encoded with respect to this new state. However, since I-frames must be compressed independently of other frames, they are much larger than P-frames, and so the I-frame is usually sent at a lower quality to mitigate the spike in bitrate. This not only affects the I-frame's quality but also the quality of subsequent frames. The overall result is that packet loss episodes can cause a jarring user experience where the video freezes followed by a period of poor quality until the codec recovers.

There are two broad techniques to combat this problem in existing systems: 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 an acceptable latency. FEC schemes send redundant packets, known as "parity" packets, to recover the lost data using traditional block codes [32, 36] or latency-optimized streaming codes [38]. However, all FEC approaches face a challenge in choosing how much redundancy to add since losses in the Internet 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 issues on subsequent frames.

In this paper, we take a different approach to loss recovery in video conferencing. We neither send redundant packets nor ask for retransmission. When a loss occurs, the receiver simply tries its best to "generate" the missing information. To do so, we build on recent advances in generative deep learning models. Generative models can reconstruct realistic images from a small amount to zero data. For example, they can reconstruct an image even if 75% of the pixels are missing [15, 27]. Unlike traditional video codecs, which use only the received data to reconstruct the source information, generative models operate like humans; they use the wealth of knowledge on how people look, dress, and move to generate missing information. For example, seeing a person's eye, they can generate the other eye, and with a glimpse of one's arm, they can generate the whole torso. Our insight is that such generative models are a natural fit for loss-resilient video conferencing. In particular, since generative models can predict missing information from their knowledge of the visual world, they can accurately generate the lost content.

We introduce Reparo, a loss-resilient generative codec for video conferencing. The design of Reparo involves two steps. In the first step, we learn how to represent the whole world of video conferencing frames using a small dictionary of visual tokens, where each token refers to a patch in a frame. The objective of this step is to capture how visual objects look and relate to each other through a small number of abstract tokens and their relationships. Reparo then operates on tokens. The transmitter uses a neural network to encode each frame into its set of tokens, which it packetizes and transmits. 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 that 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 illustrates the components of Reparo.

Beyond eliminating the need for transmitting redundant packets, Reparo offers three additional advantages. First, it can compress the video efficiently without creating dependencies between transmitted frames. This is because although Reparo does not leverage temporal relationships across frames for compression, it captures the prevalent visual features and

dependencies between objects and shapes in video conferencing frames within its codebook, which is learned ahead of time and is known to both the transmitter and receiver. During transmission, Reparo need only send token indices as opposed to the actual tokens and the underlying dependencies.

Second, traditional video codecs have a highly variable bit rate since keyframes are much larger than other frames. As a result, it is difficult for traditional video codecs to meet an exact target bitrate, and the actual bitrate will always fluctuate during transmission. In contrast, in Reparo, all frames are equal, and hence Reparo operates at a constant bitrate and can easily adapt to any target bitrate.

Third, Reparo requires only one-way communication. In past video-conferencing FEC schemes [38], the receiver sends 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. Even if one frame cannot be recovered, it will simply wait and render the next frame.

We integrate Reparo with a loss-resilient video conferencing platform [2], and evaluate Reparo and VP9+Tambur [38], a recent streaming-code based FEC approach, using the platform. Our results on a corpus of publicly available video conferencing style videos from YouTube show the following:

1. Reparo consistently improves the visual quality of displayed videos over VP9+Tambur for all loss levels. Specifically, Reparo achieves 34.1 dB, 34.0 dB, and 33.9 dB 10^{th} percentile PSNR under low, medium, and high loss levels, respectively, outperforming VP9+Tambur by 3.1 dB, 9.8 dB, and 17.9 dB.
2. Reparo nearly eliminates all video freezes, failing to render only 0.2%, 0.8%, and 2.0% of frames under low, medium, and high loss levels, respectively when VP9+Tambur fails to render significantly more frames for all loss levels (up to 29.2% of frames at high loss).
3. Reparo transmits at a constant bitrate and adapts to a range of target bitrates smoothly. In rate-limited environments, Reparo utilizes the full capacity of the link, while VP9+Tambur needs to keep the average bitrate low to avoid packet losses due to the high variability of the VP9 encoder's bitrate.

Overall, Reparo achieves superior loss-resilience and quality stability compared to existing approaches. One limitation, however, is that Reparo is more computationally intensive than traditional methods. Our current implementation requires high-end GPUs to operate in real-time. A wealth of techniques have been proposed in recent years to improve the computational efficiency of neural networks. We leave an exploration of such techniques to make Reparo accessible to a wider range of devices to future work. Nevertheless, Reparo presents a promising approach to loss-resilient video conferencing and we hope it opens up further research in this area.

2 Related Work

Video Codecs. Video applications typically use classical codecs such as VP8, VP9, H.264, H.265, and AV1 [1,4,33,39,40]. 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 relative to adjacent frames. While classical codecs are widely supported and very efficient in slow modes, in real-time video-conferencing modes, they suffer from inability to accurately match a desired target bitrate, which leads to packet loss and frame corruption when they exceed available capacity.

To overcome some of these limitations, several neural codecs have been proposed in recent years, such as Swift [8], NAS [48], LiveNAS [24], SRVC [23], Maxine [46], and DVC [31]. These codecs use an underlying low-quality video that is enhanced using a Deep Neural Network (DNN) that may be static, updated periodically, or fine-tuned online. Reparo differs from such prior neural codecs in two ways. First, some of these neural codecs still rely on temporal dependencies, wherein an undecodable or unrecoverable 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.

FEC for Loss Recovery. FEC, or Forward Error Correction, is a technique used in communication systems to recover lost data packets without retransmission. Instead of requesting the retransmission of lost packets, 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 VoIP and video conferencing, where retransmission of lost packets can cause unacceptable delays. Traditional FEC codes such as parity codes [5], Reed-Solomon (RS) codes [36], and fountain codes [32] are all block codes that are optimal for random losses, where packets are lost independently. Recently, researchers have proposed using streaming codes for FEC [38], achieving better loss recovery capabilities than block codes for bursty losses, where several packets over one or more consecutive frames are lost.

We are the first to propose a neural loss recovery scheme for video conferencing. Our scheme leverages advances in generative deep learning models to synthesize lost frames or lost patches based on its knowledge of the visual world and by conditioning on the received data.

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 that are indistinguishable from those created by humans [9,27,28,37]. These models use knowledge of the target domain to generate content under certain conditions. For example, a text gener-

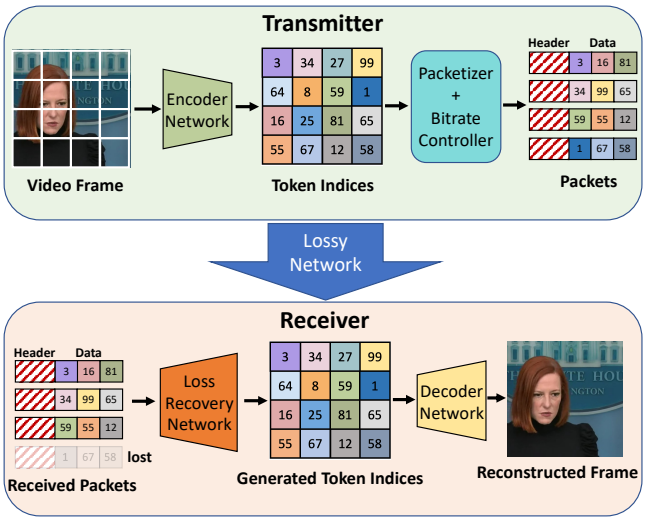

Figure 1: Overview of Reparo. An encoder-decoder pair converts between RGB image and quantized image tokens, while we introduce new modules to affect the packetization, bitrate, and loss recovery in the quantized token space to improve loss-resilience.

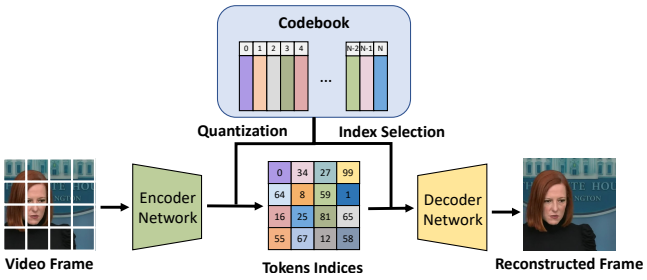

Figure 2: Token-based neural codec. The encoder converts patches from video frames into features and uses a codebook to quantize the features into tokens by finding the nearest neighbor of each feature in the codebook. The decoder then uses the tokens to reconstruct the video frame.

ative model can generate a paragraph conditioning on text prompts [9], and an image generative model can produce an image of an object using only a partial view [27].

To enhance the use of domain knowledge, many recent visual generative models have adopted a two-stage design [7, 26, 35, 44, 49]. First, they learn to represent the target domain using a visual token dictionary. Each visual token corresponds to a patch in the image and 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 [37] and image editing [27].

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 the received data, our method can generate video frames identical to the original frames, achieving loss-resilient video conferencing.

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 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 Codec

In contrast to prior work on loss-resilient video conferencing, which utilize 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 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 particular tokenizer called VQGAN [12], which

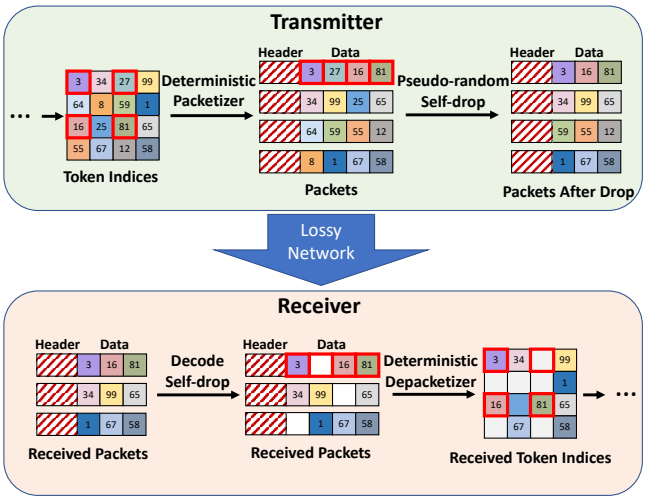

Figure 3: The transmitter first uses a deterministic packetizer to wrap image tokens into packets. Then a bitrate controller drops some tokens in each packet to adapt to the target bitrate. The receiver first decodes which tokens are dropped by the bitrate controller. It then depacketizes the received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

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 allows 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 or dictionary 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 to prepare them for transmission. The packetization strategy is designed to avoid placing adjacent tokens in the same packet since the closest tokens in the

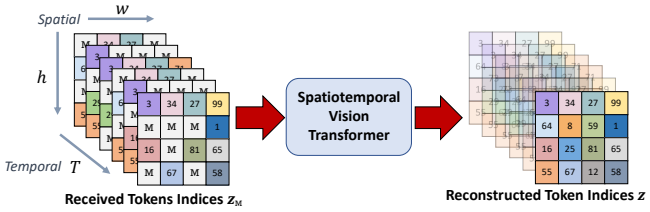

Figure 4: Loss recovery module. It uses a spatio-temporal vision transformer to generate any tokens that are lost using domain knowledge about human faces along with the received tokens in the last several frames.

image space are the most helpful for recovery when a token is lost.

In Fig. 3, the first step in the green box labelled 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 in each packet are ordered first by their row index, then by their column index in ascending order. This is just 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 that includes its frame index, packet index, and packet size so that 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 bit rate 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 bit rate 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.

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) [10]. Transformers have gained widespread popularity in computer vision and natural language processing tasks for predicting missing image patches or words [9,10,15]. The Vision Transformer 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 [3]. In each transformer block, we perform attention over the time dimension (across adjacent frames) and then over the space dimension 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 (e.g., 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 §A.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 produce the best generation of 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 at the transmitter 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 orignal frame using the codec decoder. The loss recovery module is trained by sandwiching it between the encoder, packetizer and loss simulator on one side; and the decoder on the other side, in an architecture similar to that in Fig. 1. 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,

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

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 [15]. Detailed training schemes of Reparo are included in §A.

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 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 over the codebook predicted by the loss recovery module, given the received tokens, to choose the token with the highest probability as the reconstructed token. The resulting grid of reconstructed tokens are fed into the neural decoder to generate the RGB frame for display.

4 Evaluation

We evaluate Reparo using a version of an experimental video conferencing platform called Ringmaster [2] that has support for packet loss emulation and state-of-the-art FEC schemes. We describe our baselines and experimental setup in §4.1. We evaluate Reparo under network scenarios with random packet loss in §4.2, and under packet losses induced by a rate-limited bottleneck link in §4.3. We discuss Reparo's parameter choices and latency overheads in §4.4.

4.1 Experiment Setup

Baseline. The Ringmaster platform that we use for evaluation is equipped with Tambur [38], which is a recent streaming-codes based FEC solution atop the VP9 video codec [33] that has been shown to perform better than classical block-based FEC techniques. Our benchmark for comparison is the VP9+Tambur baseline. We set Tambur's latency deadline τ to 3 frames, and its bandwidth overhead to be approximately 50%-60%. We use the same video conferencing application parameters as Tambur. The frame rate is set to 30 fps, which is typical for video conferencing.

Datasets. For training the neural codec in Reparo, we use a combination of three datasets: the FFHQ dataset (70,000 images) [21], the CelebAHQ dataset (30,000 images) [20], and the TalkingHeads dataset (∼30 hours of video) [47]. These datasets comprise high-resolution human face images and videos, making them ideal for training Reparo, which is aimed at improving video conferencing quality. We exclusively used the TalkingHeads dataset for training the loss recovery module, as this module operates on video clips instead of images.

For evaluating Reparo and the baselines in the context of video conferencing applications, we utilized a dataset that comprises publicly available videos of five YouTubers. The dataset consists of five distinct 3-minute videos corresponding to five different video conferences per YouTuber. The videos have been cropped and resized to 512×512. Each YouTuber's videos exhibit unique differences in clothing, hairstyle, accessories, or background.

Implementation. We conducted evaluations of both VP9+Tambur and Reparo using Tambur-enabled Ringmaster [38], with different bitrate constraints. To implement Reparo in the same evaluation environment, we replaced the original video codec with our neural-based codec; and the original FEC scheme with our custom packetizer, bitrate controller, and loss recovery module. The Ringmaster [2] setup reads Y4M videos at the transmitting end and displays the received videos at the receiver end. Depending on the scheme, either the VP9 encoder or Reparo's encoder is used to encode a frame. Tambur then adds FEC and packetizes the encoded frame payload while Reparo uses its customized packetizer and bitrate controller to serialize the frame into packets. Finally, at the receiver end, either the Tambur+VP9 decoder or Reparo's loss recovery module and decoder is used to decode the received packets and generate the frame. For recording displayed frames, we used the Simple DirectMedia Layer (SDL) library. We used the PyTorch Image Quality (PIQ) library [22] to compute image quality metrics.

Our neural codec and loss recovery modules were implemented in PyTorch, and they operate in real-time on three A6000 GPUs, processing 30 fps 512x512 videos. One GPU was used for the transmitter, while two GPUs were employed for the receiver. As Ringmaster and Tambur are implemented in C++, we established inter-platform-communication between Python and C++ using a socket to facilitate communication between Ringmaster and Reparo.

Metrics. We evaluate three metrics: (1) peak signal-to-noise ratio (PSNR) between the displayed and original frame, (2) percentage of non-rendered frames, and (3) latency. PSNR is computed by comparing the displayed videos to the original videos at the transmitter. Non-rendered frames are defined differently for VP9+Tambur and Reparo. For VP9+Tambur, we compute the percentage of frames that are not played by the receiver due to packet loss on that frame itself or dependency on previously undecodable frames. For Reparo, we define "non-rendered frames" as those frames with PSNR less than 30 dB,

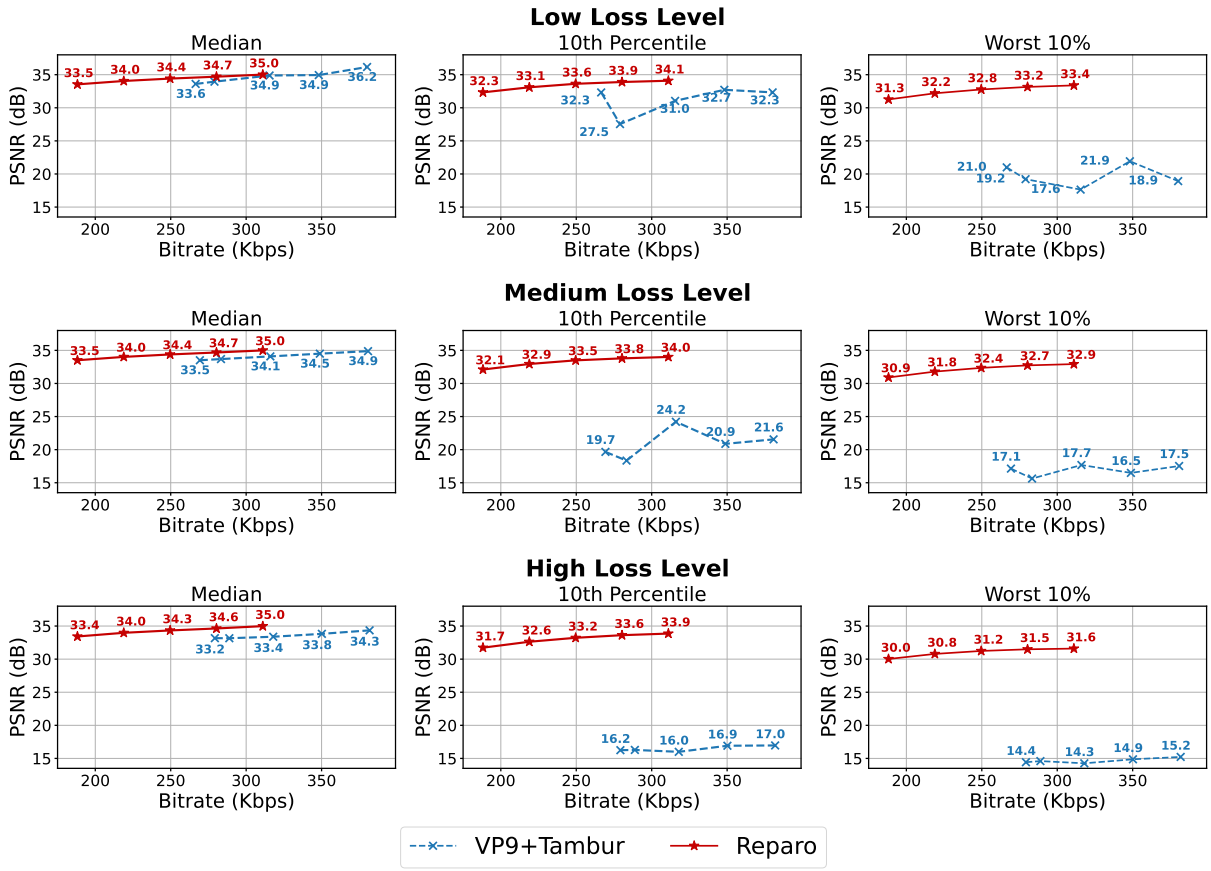

Figure 5: We report the median, 10th percentile, and worst 10% PSNR of VP9+Tambur and Reparo under different loss levels. We vary the target bitrate of Reparo and VP9+Tambur to cover different achieved bitrates. Reparo's visual quality at the tail is significantly better than VP9+Tambur across all loss levels.

as our scheme always tries to generate and render a frame. We have observed that for our dataset, VP9's PSNR rarely drops below 30 dB unless there's a frame drop, so treating Reparo's frames with PSNR below 30 dB as "non-rendered" would favor VP9+Tambur in comparisons. Non-rendered frames correlate well with standard quality-of-experience (QoE) metrics, as a large number of non-rendered frames can lead to video freezes and degrade QoE. Latency, an important metric for real-time interactivity on video conferencing applications, is evaluated by measuring the end-to-end delay between when the frame is read at the transmitter and displayed at the receiver. One-way propagation delay set to 50 ms. All metrics are aggregated over all frames of the 25 videos in our corpus, and presented as averages or distributions depending on the result. We also compute bitrate by averaging the packet sizes (without TCP/IP headers) recorded in the Ringmaster logs over the course of the entire video.

Network Scenarios. We consider two primary scenarios that can result in packet loss during transmission: (1) an unreliable network (§4.2), and (2) a rate-limited link (§4.3). In the first scenario, the network is unreliable and randomly drops packets due to poor conditions. To simulate this scenario, we use a GE loss channel that transitions between a "good" state with a low packet loss rate and a "bad" state with a high packet loss rate, similar to Tambur's setup [38]. The probability of transitioning from the good state to the bad state and vice versa is fixed at 0.068 and 0.852, respectively. The probability of loss in the good state is set to 0.04. These parameters are set to mimic Tambur's evaluation, and are computed to approximate the actual statistics over a large corpus of traces from Microsoft Teams [38]. We vary the probability of loss in the bad state to evaluate VP9+Tambur and Reparo's performance under different loss levels. Specifically, we set it to 0.25 to simulate a low loss level, 0.5 to simulate a medium loss level, and 0.75 to simulate a high loss level (the default value in Tambur's evaluation is 0.5). In the second scenario, we consider a fixed-rate link which drops packets once saturated. To simulate this, we use a FIFO queue with a fixed queue length of 6KB and a drain rate of 320 Kbps.

Reparo Parameters. In our experiments, we use a 512×512 frame size and compress it into a 32×32 tokens. With a codebook size of 1024, each token requires 10 bits to represent its

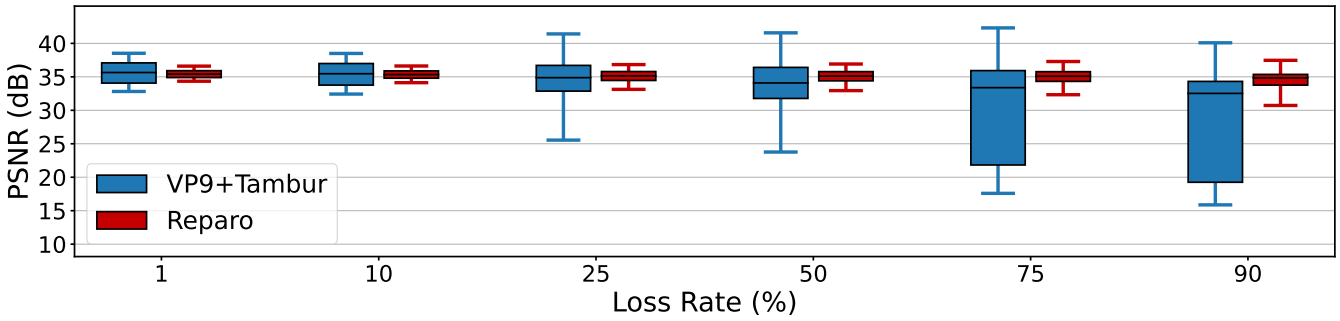

Figure 6: PSNR distribution across frames with Tambur and Reparo under different packet loss rates (for a bitrate of ∼320 Kbps). The box denotes the 25th and 75th percentile PSNR, the line inside the box denotes the median PSNR while the whiskers denote average PSNR ± 1.5×standard deviation. Reparo maintains 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.

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). 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 we show in Fig. 12.

4.2 Performance on Lossy Networks

Visual Quality. We first compare the visual quality of video displayed using VP9+Tambur and Reparo by evaluating the PSNR 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 median PSNR, 10th percentile PSNR, and the average PSNR over frames with the worst 10% PSNR. All metrics are aggregated across all frames present in our evaluation corpus. The median represents the "normal" quality of the displayed video, which occurs during "good" states of the GE loss channel. The 10th percentile and worst 10% represent the "tail" quality of the displayed video, which is affected by "bad" states of the GE loss channel.

As shown in Fig. 5, Reparo achieves higher PSNR with smaller bitrates under all settings. Specifically, under a similar bitrate (∼320 Kbps), Reparo improves the 10th percentile PSNR by 3.1 dB, 9.8 dB, and 17.9 dB under low, medium, and high loss levels, respectively. The low 10th percentile PSNR for VP9+Tambur 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 PSNR 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 PSNR 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 face images 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.

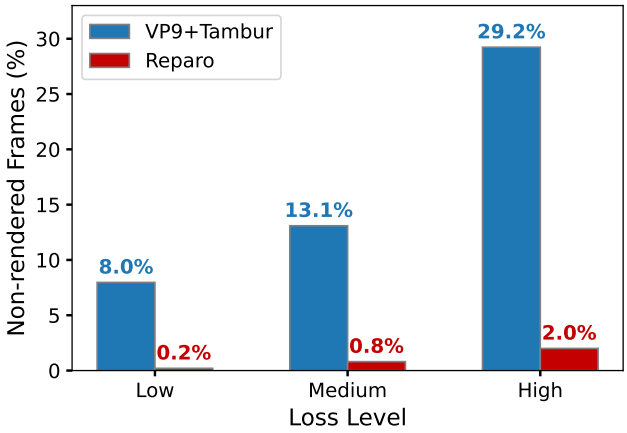

Figure 7: Comparison of percentage of non-rendered frames between Reparo and VP9+Tambur. VP9+Tambur has many more non-rendered frames than Reparo at all loss levels.

We further show the distribution of frame PSNR values across the frames in our evaluation corpus with Reparo and VP9+Tambur at ∼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

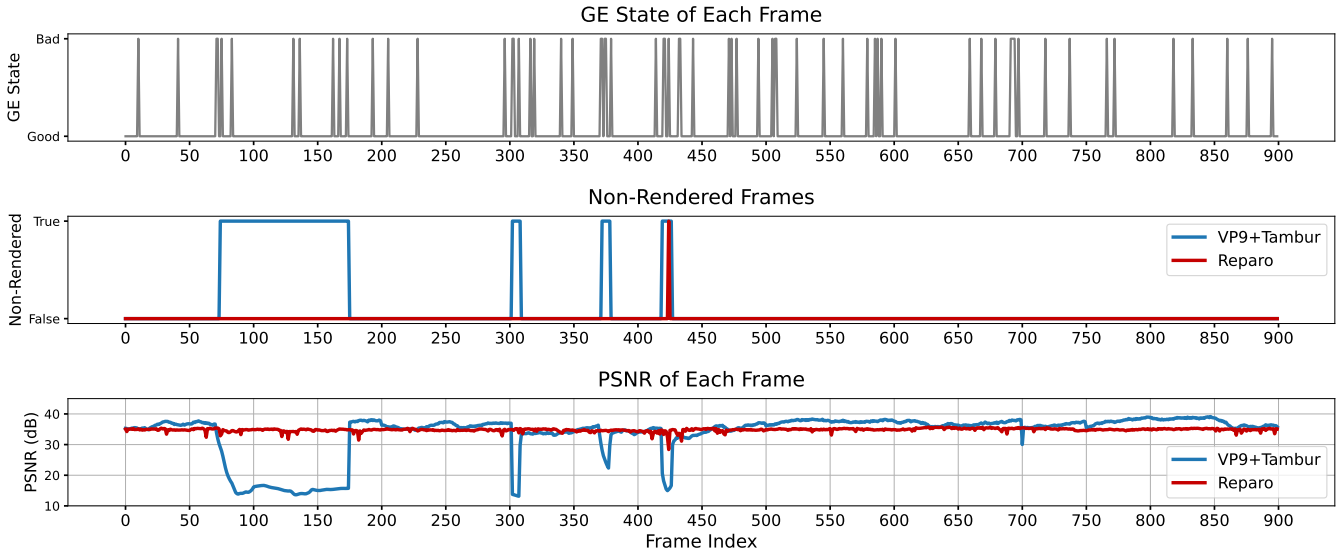

Figure 8: Timeseries comparing Tambur and Reparo on one video and loss pattern. VP9+Tambur experiences short freezes every time a set of frames are lost with a corresponding decrease in PSNR. Reparo continues rendering frames and its visual quality is a lot more stable throughout the interval.

with Reparo than VP9+Tambur, 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, VP9+Tambur is more likely to experience video freezes, resulting in more frames with low PSNR. Particularly, at a 75% packet loss, Tambur's average PSNR is 4.9 dB worse, and its distribution of frame PSNR values shows a big range: its 25th and 75th percentile PSNR values are 21.8 dB and 35.9 dB respectively. Even at a packet drop probability as low as 1%, Tambur's frame PSNR values range from 32.8 dB (5th percentile) to 38.9 dB (95th percentile). 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 VP9's PSNR rarely drops below 30 dB unless frames are lost and the video stalls. We note that such a definition favors VP9+Tambur in its comparisons with Reparo since we do not penalize VP9+Tambus for low-quality rendered frames.

We evaluated Reparo and VP9+Tambur at similar bitrates (∼320 Kbps) under various loss levels. As shown in Fig. 7, Reparo nearly eliminates non-rendered frames under all loss levels, whereas VP9+Tambur has 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.

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 VP9+Tambur over a 30-second window in Fig. 8 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 VP9+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, VP9+Tambur exhibits much lower PSNR (∼15 dB) during that timeframe between frames 72 and 177. VP9+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.

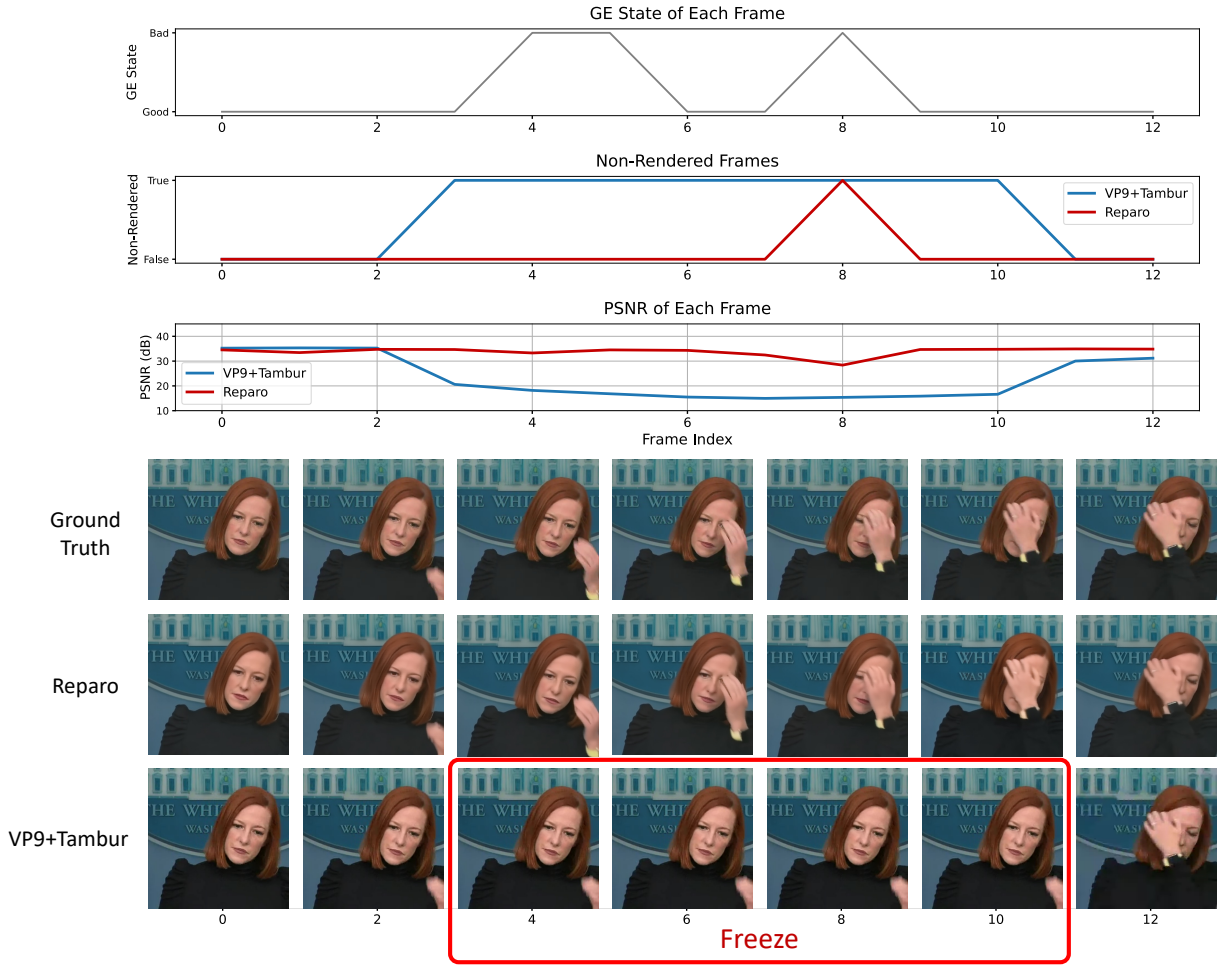

Figure 9: Qualitative results of VP9+Tambur and Reparo during a Tambur's short freeze of 8 frames. The GE loss channel is in a "bad" state at frames 4, 5, 6, and 8, causing packet losses for both VP9+Tambur and Reparo. VP9+Tambur completely freezes from frames 3 to 10 because of lost packets, leading to very low PSNR. On the other hand, though Reparo experiences the same GE loss state as VP9+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.

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 starting at frame 425.

To gain a more comprehensive understanding of the effects of packet loss events, we examine a short freeze event of VP9+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. VP9+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.

4.3 Performance on Rate-Limited Networks

In this section, we consider the packet loss caused on 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, ev-

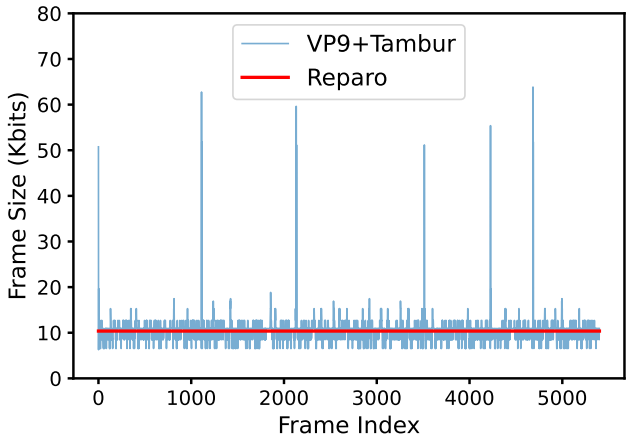

Figure 10: Per frame sizes of VP9+Tambur and Reparo for a 3 minute video. Reparo maintains the same frame size across all frames while VP9 shows variance both across adjacent predicted frames, and across periodic keyframes that are large.

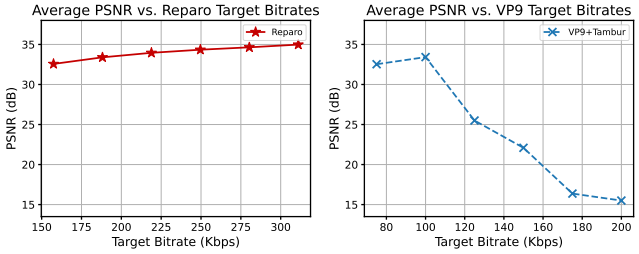

Figure 11: Average PSNR of Reparo and VP9+Tambur with different target bitrates for a fixed link capacity of 320 Kbps. Reparo's average PSNR improves as the target bitrate is increased. However, VP9+Tambur starts experiencing loss in its fixed-size queue beyond a target bitrate of 120 Kbps **due to large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

ery 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 can improve 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 [43]. Transmitted 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 VP9+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 of a keyframe across all its packets with VP9 can be much larger 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, VP9+Tambur will force another keyframe, causing the queue to remain full and preventing any frames from being transmitted, resulting in a frozen video with very low PSNR 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 [6] 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 VP9+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.

4.4 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

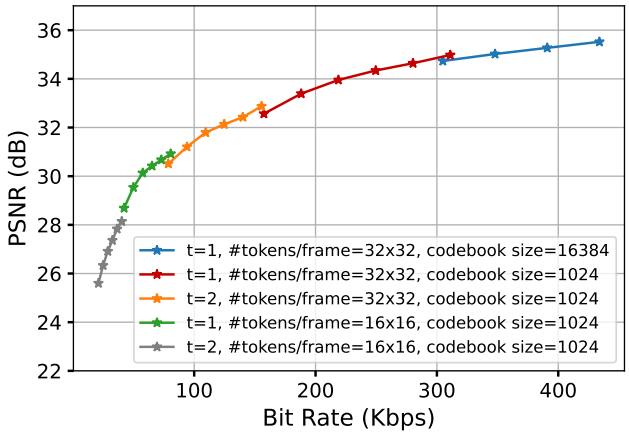

Figure 12: Variants of Reparo that operate in different bitrate regimes. Reparo achieves different bitrates by varying the number of tokens per frame, its codebook size, and the number of frames jointly encoded.

PSNR and higher bitrate due to better representational power. We can also use different codebook sizes; larger codebook sizes produce higher PSNR at the 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 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 1: 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)	9.2±0.1	0.5±0.009	35.6±1.2	15.5±0.2

Latency. In Tab. 1, 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 unoptimized implementation, the total inference delay incurred by Reparo is 60.9 ms. With typical network queuing delays of 50 ms, the end-to-end delay of Reparo is about 110 ms, which meets the industry recommendation of 150 ms for maximum tolerable latency for interactive video applications [43]. We expect that standard techniques to improve model efficiency [11,18,42] can further reduce the latency.

5 Limitations

Although Reparo offers several key advantages over traditional video codecs and existing FEC-based approaches, it also has some limitations. First, the current implementation of Reparo is based on PyTorch, and uses transformers which are computationally intensive [3, 10]. It requires high-end GPUs to operate at a reasonable speed. This limits the range of devices on which Reparo can be deployed, and the current implementation may not be suitable for low-end devices such as smartphones or tablets. However, we note that machine learning models can typically be sped up for edge device deployment using more efficient model architectures [11, 18, 34, 42], hardware design [13, 14, 51], and techniques such as knowledge distillation [17]. We leave an investigation of such optimizations to enable Reparo on edge devices to future work. Second, similar to all generative models, Reparo requires a significant amount of training data to build an accurate dictionary of visual tokens. Although we have shown that Reparo performs well on publicly available video conferencing videos, because it learns the distribution of faces and torsos typical to a video call easily, it may not generalize well to other types of videos or to videos with different resolutions and frame rates.

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 corpus of publicly available video conferencing videos and show that it consistently outperforms VP9+Tambur, a state-of-the-art loss-resilient video conferencing platform based on streaming FEC. Reparo achieves a similar PSNR as VP9+Tambur under mild loss levels and significantly improves over VP9+Tambur under heavy 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

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

[2] Ringmaster. https://github.com/microsoft/ringmaster.

[3] Anurag Arnab, Mostafa Dehghani, Georg Heigold, Chen Sun, Mario Lučić, and Cordelia Schmid. Vivit: A video vision transformer. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 6836–6846, 2021.

[4] Jim Bankoski, Paul Wilkins, and Yaowu Xu. Technical overview of VP8, an open source video codec for the web. In *2011 IEEE International Conference on Multimedia and Expo*, pages 1–6. IEEE, 2011.

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

[6] Gaetano Carlucci, Luca De Cicco, Stefan Holmer, and Saverio Mascolo. Analysis and design of the google congestion control for web real-time communication (webrtc). In *Proceedings of the 7th International Conference on Multimedia Systems*, pages 1–12, 2016.

[7] Huiwen Chang, Han Zhang, Lu Jiang, Ce Liu, and William T Freeman. Maskgit: Masked generative image transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11315–11325, 2022.

[8] Mallesham Dasari, Kumara Kahatapitiya, Samir R. Das, Aruna Balasubramanian, and Dimitris Samaras. Swift: Adaptive video streaming with layered neural codecs. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 103–118, Renton, WA, April 2022. USENIX Association.

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[10] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *Int. Conf. on Learning Representations (ICLR)*, 2021.

[11] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *The Journal of Machine Learning Research*, 20(1):1997–2017, 2019.

[12] Patrick Esser, Robin Rombach, and Bjorn Ommer. Taming transformers for high-resolution image synthesis. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12873–12883, 2021.

[13] Trevor Gale, Matei Zaharia, Cliff Young, and Erich Elsen. Sparse gpu kernels for deep learning. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–14. IEEE, 2020.

[14] Song Han, Junlong Kang, Huizi Mao, Yiming Hu, Xin Li, Yubin Li, Dongliang Xie, Hong Luo, Song Yao, Yu Wang, 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*, pages 75–84, 2017.

[15] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 16000–16009, June 2022.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[17] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[18] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[19] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017.

[20] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*, 2017.

[21] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4401–4410, 2019.

[22] Sergey Kastryulin, Jamil Zakirov, Denis Prokopenko, and Dmitry V. Dylov. Pytorch image quality: Metrics for image quality assessment, 2022.

[23] Mehrdad Khani, Vibhaalakshmi Sivaraman, and Mohammad Alizadeh. Efficient video compression via content-adaptive super-resolution. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4521–4530, 2021.

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

[25] Alina Kuznetsova, Hassan Rom, Neil Alldrin, Jasper Uijlings, Ivan Krasin, Jordi Pont-Tuset, Shahab Kamali, Stefan Popov, Matteo Malloci, Alexander Kolesnikov, 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.

[26] Doyup Lee, Chiheon Kim, Saehoon Kim, Minsu Cho, and Wook-Shin Han. Autoregressive image generation using residual quantization. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[27] Tianhong Li, Huiwen Chang, Shlok Kumar Mishra, Han Zhang, Dina Katabi, and Dilip Krishnan. Mage: Masked generative encoder to unify representation learning and image synthesis. *arXiv preprint arXiv:2211.09117*, 2022.

[28] Haohe Liu, Zehua Chen, Yi Yuan, Xinhao Mei, Xubo Liu, Danilo Mandic, Wenwu Wang, and Mark D Plumbley. Audioldm: Text-to-audio generation with latent diffusion models. *arXiv preprint arXiv:2301.12503*, 2023.

[29] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

[30] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[31] Guo Lu, Wanli Ouyang, Dong Xu, Xiaoyun Zhang, Chunlei Cai, and Zhiyong Gao. DVC: An end-to-end deep video compression framework. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 11006–11015, 2019.

[32] David JC MacKay. Fountain codes. *IEE Proceedings-Communications*, 152(6):1062–1068, 2005.

[33] Debargha Mukherjee, Jingning Han, Jim Bankoski, Ronald Bultje, Adrian Grange, John Koleszar, Paul Wilkins, and Yaowu Xu. A technical overview of VP9, the latest open-source video codec. *SMPTE Motion Imaging Journal*, 124(1):44–54, 2015.

[34] Yongming Rao, Wenliang Zhao, Benlin Liu, Jiwen Lu, Jie Zhou, and Cho-Jui Hsieh. Dynamicvit: Efficient vision transformers with dynamic token sparsification. *Advances in neural information processing systems*, 34:13937–13949, 2021.

[35] Ali Razavi, Aaron Van den Oord, and Oriol Vinyals. Generating diverse high-fidelity images with vq-vae-2. *Advances in neural information processing systems*, 32, 2019.

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

[37] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10684–10695, 2022.

[38] Michael Rudow, Francis Y Yan, Abhishek Kumar, Ganesh Ananthanarayanan, Martin Ellis, and KV Rashmi. Tambur: Efficient loss recovery for videoconferencing via streaming codes. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 953–971, 2023.

[39] Heiko Schwarz, Detlev Marpe, and Thomas Wiegand. 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.

[40] Gary J Sullivan, Jens-Rainer Ohm, Woo-Jin Han, and Thomas Wiegand. Overview of the high efficiency video coding (HEVC) standard. *IEEE Transactions on circuits and systems for video technology*, 22(12):1649–1668, 2012.

[41] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[42] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019.

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

[44] Aäron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[45] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[46] Ting-Chun Wang, Arun Mallya, and Ming-Yu Liu. One-shot free-view neural talking-head synthesis for video conferencing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10039–10049, 2021.

[47] Ting-Chun Wang, Arun Mallya, and Ming-Yu Liu. One-shot free-view neural talking-head synthesis for video conferencing. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10039–10049, 2021.

[48] Hyunho Yeo, Youngmok Jung, Jaehong Kim, Jinwoo Shin, and Dongsu Han. Neural adaptive content-aware internet video delivery. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 645–661, 2018.

[49] Jiahui Yu, Xin Li, Jing Yu Koh, Han Zhang, Ruoming Pang, James Qin, Alexander Ku, Yuanzhong Xu, Jason Baldridge, and Yonghui Wu. Vector-quantized image modeling with improved vqgan. *arXiv preprint arXiv:2110.04627*, 2021.

[50] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 586–595, 2018.

[51] Zhekai Zhang, Hanrui Wang, Song Han, and William J Dally. Sparch: Efficient architecture for sparse matrix multiplication. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 261–274. IEEE, 2020.

A Implementation Details

Because of GPU memory limitation, we adopt a two-stage training recipe for Reparo similar to many prior approaches [12, 27]. 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.

A.1 Neural Codec

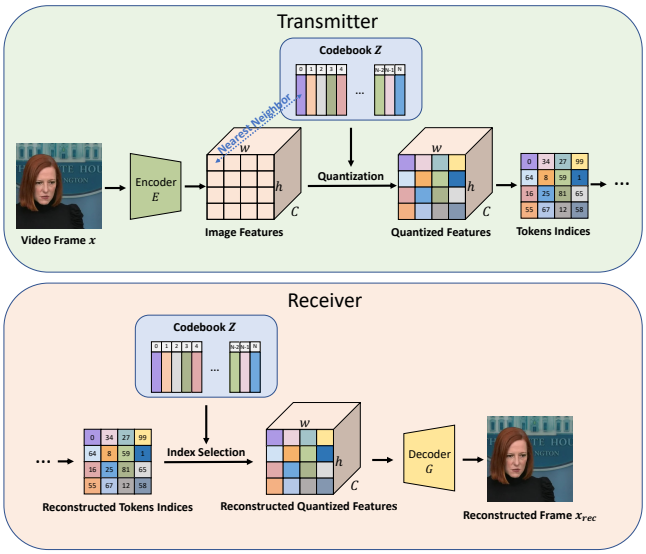

Figure 13: Token-based neural codec. The transmitter first uses the encoder to convert patches from video frames into features. It then uses a codebook to quantize the features into tokens by finding the nearest neighbor of each feature in the codebook. The receiver first maps the received and generated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

Model Structure. We use a CNN-based VQGAN [12] encoder and quantizer to tokenize the $3 \times 512 \times 512$ input frame to $128 \times 32 \times 32$ 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 $3 \times 512 \times 512$ video frame. The encoder consists of 5 blocks and each block consists of 2 residual blocks which follow standard ResNet's residual block design [16]. 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.

Table 2: VQGAN codec training setting.

Parameter	Value
Optimizer	Adam [30]
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 [29]
Warmup Epochs	0
Gradient Clip	0
Dropout	0

Training schemes. We follow the original VQGAN training recipe [12] 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 [50] between the input and reconstructed frame, and a discriminative loss [19] between the input and reconstructed frame. Detailed descriptions of the losses can be found in the VQGAN paper [12].

We use the officially released VQGAN encoder and decoder pre-trained on OpenImages [25] 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. 2 for the training recipe of our VQGAN codec.

A.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 [10], which consists of a stack

Table 3: Loss recovery module training setting.

Parameter	Value
Optimizer	Adam [30]
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 [29]
Warmup Epochs	10
Training Epochs	200
Gradient Clip	3.0
Label Smoothing [41]	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

of spatio-temporal Transformer blocks [45]. 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. 3 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.

A.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 bitrate when transmitting all tokens is 300 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 $4 \times \text{frame index} + \text{packet index}$. 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.