Towards

Robust And Practical Neural Video-Conferencing

by

Vibhaalakshmi Sivaraman

B.S.E., Princeton University (2017)
S.M., Massachusetts Institute of Technology (2019)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

Massachusetts Institute of Technology

February 2024

Authored by: Vibhaalakshmi Sivaraman
Department of Electrical Engineering and Computer Science
December 15, 2023

Certified by: Mohammad Alizadeh
Associate Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by: Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Towards Robust And Practical Neural Video-Conferencing

by

Vibhaalakshmi Sivaraman

Submitted to the Department of Electrical Engineering and Computer Science
on December 15, 2023, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

Video conferencing systems suffer from poor user experience when network conditions deteriorate because current video codecs cannot operate at extremely low-bitrates or under lossy network conditions without frame corruption or video freezes. To tackle the low-bitrate problem, several neural alternatives have been proposed that reconstruct talking head videos using sparse representations of each frame such as facial landmark information. However, these approaches produce poor reconstructions in scenarios with major movement or occlusions over the course of a call, and do not scale to higher resolutions. To cope with packet loss, most systems use retransmissions or Forward Error Correction (FEC) techniques. Retransmissions are impractical in real-time settings due to their slow turnaround times while Forward Error Correction (FEC) techniques require extensive tuning to ensure the right level of redundancy. Instead, this dissertation develops a new paradigm for video conferencing using a suite of generative techniques based on super-resolution and attention mechanisms to improve video conferencing experience across both classes of poor network conditions.

First, we present Gemino, a new neural compression system for video conferencing based on a novel *high-frequency-conditional super-resolution* pipeline. Gemino upsamples a very low-resolution version of each target frame while enhancing high-frequency details (e.g., skin texture, hair, etc.) based on information extracted from a single high-resolution reference image. Such a design overcomes the robustness issues of models that rely on only facial landmarks under extreme motion. Gemino's design includes a multi-scale architecture that runs different components of Gemino at different resolutions, allowing it to scale to resolutions comparable to 720p. We also personalize the model to learn specific details of each person, achieving much better fidelity at low bitrates. We implement Gemino atop *aiortc*, an open-source Python implementation of WebRTC, and show that it operates on 1024×1024 videos in real-time on a Titan X GPU, and achieves 2.2–5× lower bitrate than traditional video codecs for the same perceptual quality.

Since Gemino is not designed to leverage high-resolution information from multiple references, we further design Gemino (Attention), a version of Gemino that computes "attention" or a weighted correspondence between regions of different reference frames and the target frame. This attention design is in contrast to the optical flow framework within Gemino that is restricted to merely linear translations from regions of a single reference frame to its target region. Such an attention-based design is, instead, able to combine information across different references and use the best parts of each reference frame to improve the fidelity of the reconstruction.

Lastly, we develop Reparo, a loss-resilient generative codec for video conferencing that reduces the duration and impact of video freezes during outages. Reparo's compression does not depend on temporal differences across frames, making it less brittle in the event of packet loss. Reparo automatically generates missing information when a frame or part of a frame is lost, based on the data received so far, and the model's knowledge of how people look, dress, and interact in the visual world.

Together, these approaches suggest an alternate future for video conferencing powered by neural codecs that can operate in extremely low-bandwidth scenarios as well as under lossy network conditions to enable a smoother video conferencing experience.

Thesis Supervisor: Mohammad Alizadeh
Title: Associate Professor of Electrical Engineering and Computer Science

To my dearest paati, Rajalakshmi Ramamurti.

Acknowledgments

Mohammad Alizadeh has been the kindest and most patient advisor I could have ever asked for. His attention to detail and commitment to perfection are unmatched; he does not relent until he fully understands every aspect of a project down to its most minute details. I owe him a lot for being very supportive of my wanting to switch research directions relatively late in my PhD, and then, for encouraging many lab-mates to help with the cause. I am arguably a lot less afraid of exploring and developing expertise in new areas thanks to him.

My thesis committee has played a crucial role in shaping this dissertation and my PhD experience. Vivienne Sze provided much needed expertise on video codecs as well as model optimization without which many of the speedups we achieved would not have happened. Frédo Durand practically adopted me into his group and helped me familiarize myself with the computer vision vocabulary. Hari Balakrishnan has provided me with very timely personal and research advice over the years. Most notably, he suggested I move to the US for undergrad, pick MIT for graduate school, and switch research agendas in my fourth year. Each of these decisions has had a significant impact on this dissertation.

My student researcher engagement with Google in the last year of my PhD taught me to appreciate the nuances of industry and academic research. I am thankful for all the time that Rahul Garg, Anne Menini and Xuan Luo invested into my project, and for the willingness on everyone's part to find a way to include it in this dissertation.

I have been blessed with many mentors, formal and informal, over the years who have played different roles and filled various needs. Srinivas Narayana taught me how to set up and run experiments, and how to describe the results in a paper. Shaileshh Venkatakrishnan taught me to not be afraid of math and to keep iterating until a talk is perfectly clear. Jennifer Rexford, my undergraduate research advisor, played a pivotal role in my decision to pursue a PhD, setting a gold-standard with her exemplary teaching, research and service to department. Irene Greif has humbled me with the time and thought she has invested into my success as a PhD student, and her absolute willingness to help me no matter what the situation is. Giulia Fanti's constant reminders that I was a capable and creative researcher with plenty of ideas were much needed during the initial years of my PhD.

I am equally fortunate to have had the opportunity to mentor many younger students who have taught me a lot about managing others. Kathy Ruan taught me how to break code requirements down to the very basics. This dissertation would not have been possible without Pantea Karimi and Vedantha Venkatapathy, and their extreme patience with me. Watching them both mature into independent researchers has been extremely rewarding. I deeply admire Caroline Jin's ability to hack together prototypes in short time-spans with nothing more than a gentle nudge from me.

I have been fortunate to work with many collaborators — Ravichandra Addanki, Mohammad Alizadeh, Frédo Durand, Lijie Fan, Giulia Fanti, Sadjad Fouladi, Rahul Garg, Prateesh Goyal, Caroline Jin, Tianhong Li, Xuan Luo, Pantea Karimi, Dina Katabi, Mehrdad Khani, Anne Menini, Radhika Mittal, S. Muthukrishnan, Srinivas Narayana, Vikram Nathan, Parimarjan Negi, Jennifer Rexford, Ori Rottenstreich, Kathleen Ruan, Vivienne Sze, Weizhao Tang, Shaileshh Venkatakrishnan, Vedantha Venkatapathy, Pramod Viswanath, Lei Yang — who I have really enjoyed working with and learnt a lot from.

Thank you to my G982 officemates, NMS labmates, and G9 cohabitants, past and

present —Venkat Arun, Arjun Balasingam, Frank Cangialosi, Inho Cho, Prateesh Goyal, Pouya Hamadanian, Songtao He, Pantea Karimi, Mehrdad Khani, Moein Khazraee, Sunghyun Kim, Derek Leung, Chenning Li, Alex Mallery, Hongzi Mao, Radhika Mittal, Akshay Narayan, Srinivas Narayana, Arash Nasr-Esfahany, Vikram Nathan, Pari Negi, Ravi Netravali, Amy Ousterhout, Seo Jin Park, Deepti Raghavan Sudarsanan Rajasekaran, Ahmed Saeed, Harsha Sharma, Will Sussman, Lily Tsai, Shaileshh Venkatakrishnan, Frank Wang, Lei Yang, Migran Yang, and Zhizhen Zhong—for fostering a very collegial environment in Stata that made me look forward to coming in. I cherish the many conversations, often untimely ping-pong sessions, Tosci's and coffee trips, and lab outings. I am also thankful for many of my officemates' willingness to share their Vim bindings, iTerm settings, Git configs, and R scripts that made bootstrapping significantly smoother for me.

I owe a lot to the administrative staff at MIT for making my PhD a smooth experience. Thanks to Sheila Marian, Angelly Arriola, Janet Fischer, Alicia Duarte, Leslie Kolodziejski, Sylvia Heistand, and CSAIL TIG for responding to all my queries and concerns.

The last six years in Boston would not have been enjoyable without my friends. Pritpal Singh Kanhaiya, Nalini Singh, Yamin Arefeen, Aniruddh Raghu and Rachel Holladay have been beyond supportive during some of the hardest patches of the PhD. I will cherish the memories of countless walks and runs around the Charles river with many of them. My Princeton friends gave me a sense of home in the US without which I would not have survived my PhD. Special thanks to Pallavi Koppol and Corrie Kavanaugh for always bringing a smile to my face no matter the situation.

My extended family has been a great source of comfort and warm food throughout the PhD. My in-laws have been supportive in every way possible, particularly by housing me through the first few months of the COVID-19 pandemic in the middle of my PhD.

My late grandfather, Dr. V. Ramamurti, instilled a deep respect for academic institutions and a love for intellectual curiosity. He was not around to see me get into graduate school or finish a PhD, but I hope he is proud of this dissertation. My grandmother, Rajalakshmi Ramamurti, or Paati has been my number one cheerleader. She was neither encouraged to speak her mind nor allowed to pursue the music career she aspired for. Yet, she raised a very strong daughter and granddaughter that she wholeheartedly supported in all their endeavors. I dedicate this dissertation to her in honor of the inspiration she is.

My parents, Rama and Sivaraman, have taught me to always do my best, no matter what I pursue. They have given me all the emotional and financial support I have ever needed, never once considering the cost to themselves. This dissertation would not have happened if not for them. My brother, Anirudh Sivaraman, has been a role model to me since day one. My childhood strategy of mimicking everything he did carried me all the way to a PhD in networking from MIT. I am grateful for his willingness to patiently listen to my many rants over the years. Lastly, I owe a lot to my husband, Arjun Venkataraman, for supporting every step of my PhD journey and committing to six years of long distance. I am thankful for his ability to ask me the hard questions, and for reminding me that there is more to life than work.

Previously Published Material

Chapter 3 revises a conference paper to appear at USENIX NSDI 2024 [134]: Vibhaalakshmi Sivaraman, Pantea Karimi, Vedantha Venkatapathy, Mehrdad Khani, Sadjad Fouladi, Mohammad Alizadeh, Frédo Durand, Vivienne Sze. Gemino: Practical and Robust Neural Compression for Video Conferencing.

Chapter 4 revises a technical report [135]: Vibhaalakshmi Sivaraman, Xuan Luo, Anne Menini, Mohammad Alizadeh, Rahul Garg. Multi-Resolution Multi-Reference Talking Head Synthesis via Implicit Warping.

Chapter 5 revises another pre-print [92]: Tianhong Li, Vibhaalakshmi Sivaraman, Lijie Fan, Mohammad Alizadeh, Dina Katabi. Reparo: Loss-Resilient Generative Codec for Video Conferencing. Reparo: Loss-Resilient Generative Codec for Video Conferencing.

Contents

List of Figures

List of Tables

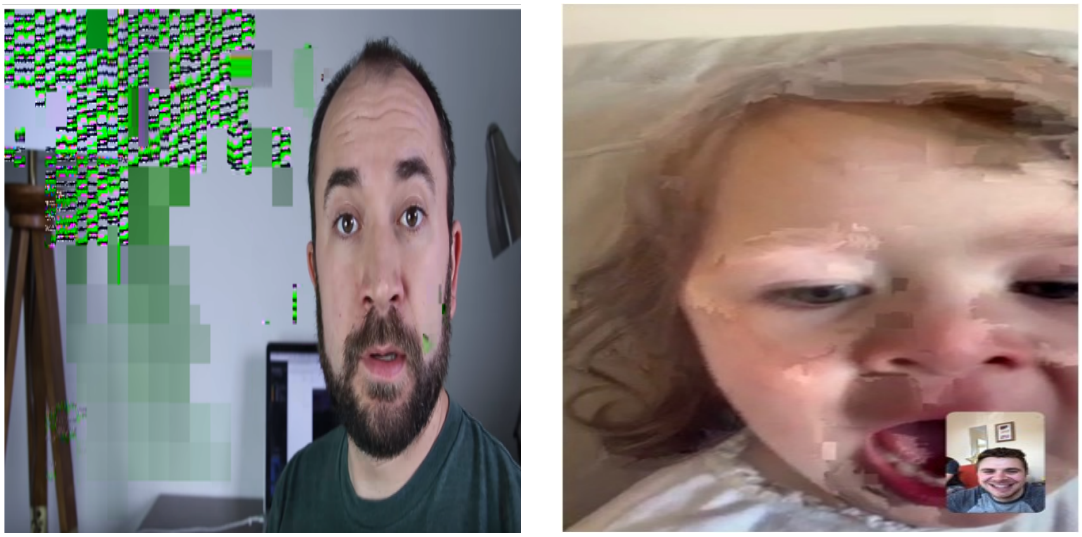
Chapter 1

Introduction

1.1 Motivation

Video conferencing applications have become a crucial part of modern life. Though video calling has been in use for personal use cases such as long-distance family calls for many decades, the technology has become particularly integral for the smooth functioning of all aspects of society since the COVID-19 pandemic. Video conferencing is the reason for how normalized telehealth, remote and hybrid work, virtual educational and vocational options, virtual tours and virtual fitness classes have become lately. This proliferation of use cases is reflected in the growth trends of video calling services through the last few years. Through 2020 and 2021, the number of monthly active users of popular applications like Zoom, Microsoft Teams, and Google Meet grew to over 21 times of their pre-pandemic levels [23]. Though travel and in-person work have returned to their pre-pandemic levels in large parts of the world, the video conferencing market is still expected to continue to grow over the next decade [1].

However, today's systems continue to suffer from poor user experience: in particular, poor video quality and unwelcome disruptions are all too common. The issues shown in Fig. 1-1 are very familiar to all of us. Fig. 1-1(b) shows an example of a pixelated screen resulting from low resolutions during poor network conditions, while Fig. 1-1(a) shows glitches in the displayed frame due to lost packets. Situations such as these arise because of the inability of today's applications to operate in low-bandwidth or lossy network scenarios. For instance, Zoom recommends a minimum bandwidth of 1.2 Mbps for one-on-one meetings and 2-3 Mbps for group meetings [29]. In certain parts of the world, household Internet broadband speeds remain far below these recommendations for reliable video conferencing. As seen in Fig. 1-2, large swaths of the population in Africa and Asia had average Internet broadband speeds less than 10 Mbps in 2022 [28], with the five slowest countries having speeds under 1 Mbps. Mobile bandwidth is even more restricted: SpeedTest's Global



(a) Glitches in displayed frame from packet loss. (b) Pixelation effects in a low-resolution frame.

Figure 1-1: Typical issues encountered during a video call during poor network conditions.

Index [13] suggest that global mobile bandwidth average is 50% of broadband speeds. This means that many of these countries cannot support video calls for even one person in a household, much less for an entire household. Even in regions of North America and Europe with high broadband speeds [28], over 30% of users surveyed about their video conferencing experience claimed that "video quality" issues were their biggest pain point [20]. The same survey indicates that over 50% of users deem audio issues their biggest concern. Often, turning off video often leads to better audio quality [22]; this suggests that most frustrations with video conferencing applications come from the same root cause of insufficient capacity for video.

The aforementioned issues arise because the user experience is not just determined by the average bandwidth, but rather by tail events of low bandwidth and outages (a few seconds every 5–10 minutes) that cause glitches and disrupt the video call. When the network deteriorates, even briefly, existing video conferencing solutions cope to an extent by lowering quality, but below a certain bandwidth (e.g., 100s of Kbps for HD video), they must either suspend the transmission altogether or risk packet loss and frame corruption. Further, since video frames are encoded based on *delta encoding* or the differences between frames, one corrupt frame often causes an extended outage due to dependencies across frames.

1.2 Existing Approaches

1.2.1 Achieving Low Bitrates

Video applications typically use classical codecs such as VP8, VP9, H.264, H.265, and AV1 [34, 109, 129, 136, 7]. These codecs compress video frames using block-based motion

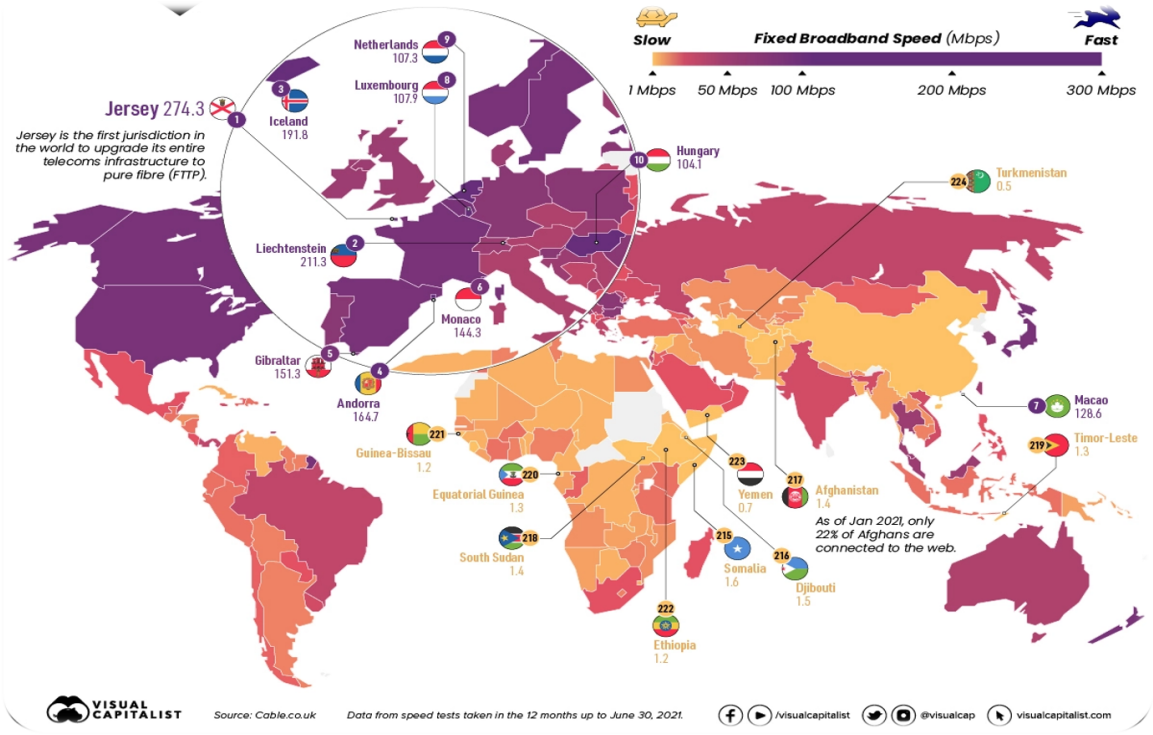


Figure 1-2: Internet broadband speeds in the world. Many populous parts of the world have broadband speeds well under 10 Mbps, with the slowest countries having less than 1 Mbps.

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. These codecs, though widely supported, rely on search-based algorithms that compare every block in the current frame to a past frame. Consequently, these codecs are very efficient in slow modes, but are constrained by search speed in real-time video-conferencing modes. They rarely achieve bitrates lower than few hundreds of Kbps, and also do not accurately match a desired target bitrate, which leads to packet loss and frame corruption when they exceed available capacity.

Recently, several neural approaches for face image synthesis have been proposed that deliver extreme compression by generating each video frame from a sparse representation (*e.g.,* keypoints) [132, 149, 162, 113, 143]. These techniques have the potential to enable video conferencing with one to two orders of magnitude reduction in bandwidth (as low as $\sim$10 Kbps [149, 113]) when transmitting the sparse representations alone, but their lack of robustness and high computational complexity hampers their practicality. Specifically, synthesis approaches work by "warping" a reference image into different target poses and orientations captured by such sparse keypoints. These methods produce good reconstructions when the difference between the reference and the target image is small, but

they fail (possibly catastrophically) in the presence of large movements or occlusions. In such cases, they produce poor reconstructions, for both low-frequency content (*e.g.,* missing the presence of a hand in a frame altogether) and high-frequency content (*e.g.,* details of clothing and facial hair). As a result, while synthesis approaches show promising average-case behavior, their performance at the tail is riddled with inconsistencies in practice. Furthermore, real-time reconstruction is only feasible at low resolution for such models [132, 113], even on high-end GPUs, while typical video conference applications are increasingly designed for HD, Full HD, and even 4K videos [31, 30, 6]. Naïvely reusing these models on larger input frames can quickly become prohibitively expensive as the resolution is increased.

1.2.2 Tackling Packet Loss

Another issue with traditional codecs is that the interdependence between frames makes them brittle under packet loss. For example, P-frames are encoded with reference to previous frames, and to decode them, the receiver must first decode the frames on which it depends. If prior frames cannot be decoded due to lost packets, the receiver stalls and cannot proceed, causing the video to freeze. Once in this state, the receiver requests a retransmission and can only decode the relevant frame once it has received the missing data. If the data continue to be missing, the receiver requests a new key frame or I-frame to effectively reset the encoder state so that any future frames are encoded based on this new state that is corruption-free. This works because I-frames are compressed independently of other frames; any loss of prior frames does not affect an I-frame's decodability. However, as a result of this independent compression, I-frames are much larger than P-frames, and sending a large frame during or just after poor network conditions means that the I-frame will have really poor quality, which impacts subsequent frames' quality. 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.

The typical way to overcome packet loss in existing video systems is to use either retransmissions or forward error correction (FEC). However, since real-time applications have latency requirements that dictate how soon lost packets need to be retrieved, retransmission is only used if the round trip time is short. The more common approach is to use FEC wherein the sender sends redundant "parity" packets along with the standard video stream so that the receiver can recover any lost packets using traditional block codes [120, 104] or latency-optimized streaming codes [126]. FEC approaches are optimized to balance redundancy and efficiency based on the expected loss rate under typical network conditions. This is important because sending too many parity packets can be wasteful, but sending too few may make the video stream much less resilient to losses. Tuning the

level of redundancy perfectly for unpredictable network conditions is hard. Though recent years have seen an explosion of neural alternatives for video compression and transmission such as Swift [49], NAS [158], LiveNAS [84], SRVC [83], Maxine [149], and DVC [101], they often still rely on temporal dependencies or have poor reconstruction quality under certain tail scenarios that makes them unsuitable for video conferencing under lossy conditions.

1.3 Key Contributions

This thesis describes two classes of neural codecs that independently target (1) low-bandwidth scenarios and (2) lossy network conditions. We briefly list the key contributions, and then describe them in more detail.

- Gemino, a new neural compression system for video conferencing that uses a novel high-frequency conditional super-resolution pipeline to operate at audio-like bitrates.
- Gemino (Attention), an alternate design for the high-frequency conditional super-resolution pipeline that leverages attention instead of optical flow, allowing it to scale to multiple reference images.
- Reparo, a loss-resilient generative codec that reduces video freezes during network outages.

1.3.1 Gemino: A Robust Low-Bitrate Neural Codec

Chapter 3 proposes Gemino, a neural compression system for low-bitrate video conferencing, designed to overcome the robustness and compute complexity challenges associated with keypoint-based techniques. Gemino targets extreme compression scenarios, such as delivering video in ∼100 Kbps or less. At such bitrates, the bandwidth required for video becomes comparable to a typical audio call [27], greatly expanding the range of networks that can support video conferencing.

Gemino design. We begin with the observation that current synthesis approaches, in an effort to squeeze the most compression, overly rely on keypoints, a modality with limited semantic information about the target frame. This causes several inevitable failures (§3.1). For example, if a user's hand is absent in the reference frame but appears in the target, there is no way to reconstruct the hand by warping the reference image. We would need to send a new reference frame that includes the hand, but sending a high-resolution frame (even occasionally) incurs significant cost. Instead, Gemino directly transmits low-resolution video, which includes significantly more information about the target frame, and upsamples it to the desired resolution at the receiver. Using low-resolution video is viable because modern codecs [46, 110, 35, 130] compress them very efficiently. For example, sending

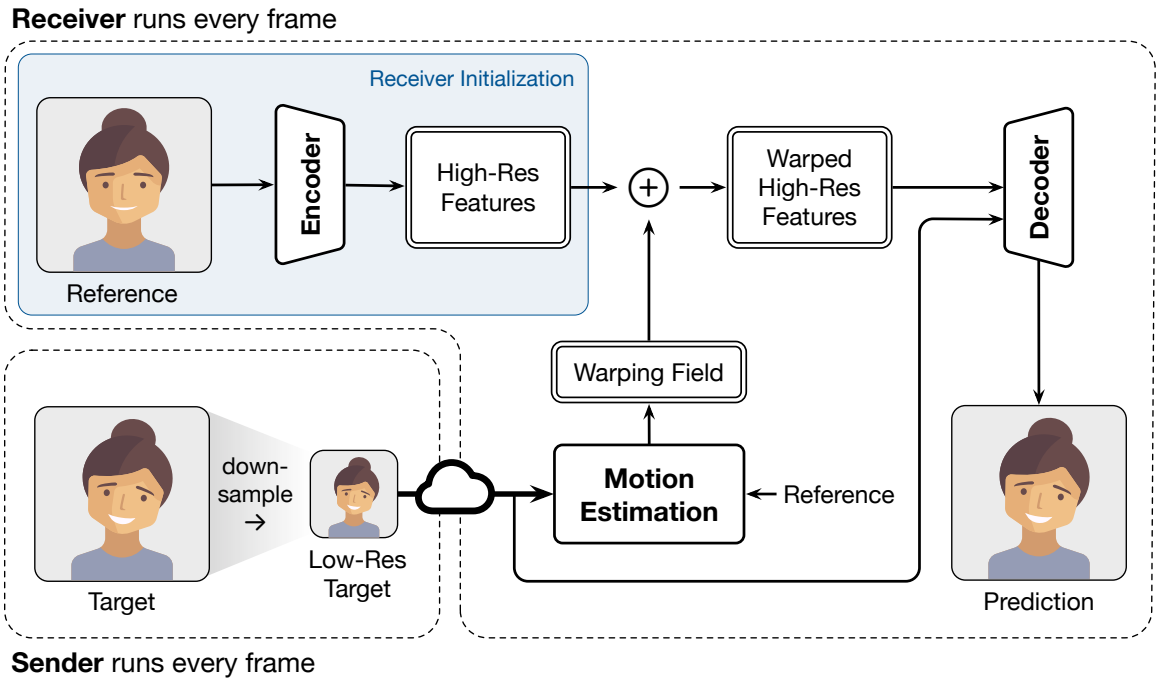


Figure 1-3: Gemino's design. The sender sends a downsampled version of the target frame across the network to the receiver that has a reference image of the speaker in the current video conference setting. The encoder network encodes and warps features from the reference image based on the motion estimated between the reference and target frames. The decoder network at the receiver upsamples the downsampled frame with help from the warped encoded reference features.

128×128 resolution video in our implementation consumes ∼15 Kbps, only slightly more than would be needed to transmit keypoints [149, 113]. We posit that the robustness benefits of providing the receiver with more information for reconstruction far outweigh the marginal bandwidth cost.

It is challenging to upsample a video significantly while reconstructing high-frequency details accurately. For example, at 1024×1024 resolution, we can see high-frequency texture details of skin, hair, and clothing, that are not visible at 128×128 resolution. To improve high-frequency reconstruction fidelity, Gemino uses a reference frame that provides such texture information in a different pose than the target frame. Like synthesis approaches, it warps features extracted from this reference frame based on the motion between the reference and target frames, but it combines it with information extracted from the low-resolution target image to generate the final reconstruction (Fig. 1-3). We call this new approach *high-frequency-conditional super-resolution*.

Compute and Fidelity Optimizations. Gemino uses several further optimizations to improve reconstruction fidelity and reduce computation cost. First, we *personalize* the model by fine-tuning it on videos of a specific person so that it can learn the high-frequency

detail associated with that person (e.g., hair, skin wrinkles, *etc.*). Personalizing the model makes it easier to transition texture information from the reference to the target frame. Second, we train the neural network using decompressed low-resolution frames obtained from a standard codec so that it learns to reconstruct accurate target frames despite codec-induced artifacts. We design Gemino to work with any starting resolution so that it can achieve different rate-distortion tradeoffs based on the available network bandwidth. The resolution at a particular bitrate is chosen by profiling standard codecs and picking the highest resolution that can achieve that bitrate.

Lastly, as the target resolution increases, it is essential to reduce the number of operations required per-pixel for synthesis. Otherwise, the compute overheads of running neural networks at higher resolutions become prohibitively high. To achieve this compute reduction, we design a *multi-scale* architecture wherein different parts of the model operate at different resolutions. For instance, the module that produces the warping field uses low-resolution versions of the reference and target images, while the encoder and decoder networks[1] operate at full resolution but are equipped with additional downsampling blocks to reduce the operations per pixel. This design allows us to obtain good reconstruction quality while keeping the reconstruction real time. The multi-scale architecture will be particularly salient as we transition to higher resolution video conferencing applications in the future. We also further optimize our model using neural architecture search techniques such as NetAdapt [156] to reduce the compute footprint.

Results. We implement and evaluate Gemino within *aiortc* [3], a Python implementation of WebRTC, and show the following:

1. Gemino achieves a perceptual quality (LPIPS) [163] of 0.21 at 105 Kbps, a 5× and 2.2× reduction from VP8 and VP9's default Chromium and WebRTC settings respectively.
2. In lower bitrate regimes where current video conferencing applications *cannot* operate, Gemino outperforms bicubic upsampling and SwinIR [94] super-resolution from compressed downsampled VPX [2] frames. Gemino requires 50 Kbps to achieve an LPIPS of 0.23, a 2× and 3.9× reduction compared to bicubic and super-resolution.
3. Our model transitions smoothly across resolutions, and between generated and synthetic video, achieving different rate-distortion tradeoffs based on the target bitrate.
4. Our model benefits considerably (~ 0.04 in LPIPS) from personalization and including the VPX codec at train-time ($\sim$0.05 in LPIPS). Our optimizations atop the multi-

[1] "Encoder" and "decoder" throughout this dissertation refer to the neural network pair typically used in GANs [59]. When we specify *VPX* encoder or decoder, we refer to the video codec's encoder and decoder.

[2] VPX refers jointly to VP8 and VP9; we delineate them when necessary.

scale architecture enable real-time inference on 1024×1024 frames on a Titan X GPU.

1.3.2 An Attention-Based Design for Gemino

Gemino [134] uses a single reference frame in its reconstruction pipeline for synthesizing high-resolution frames in a target pose. A natural question is whether using more references can improve the reconstruction fidelity. However, Gemino, like other novel-view synthesis approaches [162, 132, 149], leverages optical flow. Specifically, Gemino encodes features from the reference frame, warps them based on the optical flow estimate, and then decodes the warped features. The optical flow, measured in two-dimensions, captures which reference feature (or pixel) needs to be moved to a given target image location in order to produce the reconstruction. To extend this approach to multiple reference frames, Gemino would have to encode multiple source or reference images, independently warp each of the features based on their respective optical flows relative to the target, and utilize an additional layer [144] to compute how to combine the different optical flows before the decoding step. Such flow estimation can be challenging, particularly when there is extreme motion, and image or feature correspondence is minimal.

A more versatile alternative to optical flow is attention [142, 51]. The basic approach within an attention layer is to compute correspondence between *query* and *key* features using a dot product and then, extract the corresponding *value* features for those keys that have highest correlation with the query. In our context, the attention layer identifies which spatial locations in the target frame (query) should draw upon which locations in the reference (key). While the (low-resolution) key features are used for computing correspondence with the query, each key feature maps to a corresponding high-resolution *value* feature which is decoded after the attention operation. Attention is similar to optical flow in that it identifies a correspondence between source and target pixels. However, it is much more general in that it allows you to produce a weighted combination of input locations for every output location, rather than rely on a single input location that needs to be moved to a target location. This versatility comes at the cost of higher computation [51, 167] but provides much more flexibility to the model in the form of a larger corpus of features, potentially from multiple references [144], to pick from. It also opens possibility of aggregating a set of person- or video-specific features at train-time that the attention mechanism can then learn to draw upon appropriately in the inference pipeline.

In Chapter 4, we discuss an alternate design for Gemino that uses attention mechanisms instead of optical flow based models. Specifically, Gemino (Attention) encodes a low-resolution target image as well as low and high-resolution versions of the reference into a set of features. It then computes attention or correspondence between the features from

the low-resolution target (query features) and the low-resolution reference (key features), and extracts the corresponding high-resolution reference features (value features). The attended features from the high-resolution reference features are then put through a series of decoding layers to produce the final reconstruction. This attention structure naturally lends itself to multiple reference frames; we simply extract key-value pairs from each reference frame's low-resolution and high-resolution versions and stack them together in the attention pipeline. Note that the model is trained once for a single reference image, but extends at inference time to multiple reference images. This is because attention is merely
more keys than it was originally trained for without retraining.

We evaluate Gemino (Attention) on publicly available TCDTimit [63] dataset and show that our novel multi-resolution attention design provides an improvement of 1.71 dB in PSNR, 1.13 dB in SSIM and 0.06 in LPIPS over a similar design that runs attention only at the coarsest level. Further, the same model when evaluated using five references provides an additional improvement of 0.15 dB in PSNR, 0.07 dB in SSIM, and 0.01 in LPIPS. Since TCDTimit [63] lacks sufficient variation in pose and backgrounds, we also evaluate on a more diverse dataset and observe that we get an improvement of nearly 1 dB in both SSIM and PSNR, and a 20% decrease (0.01) in LPIPS when using 10 references instead of 1. As the number of reference frames grows, our model incurs higher computational costs relative to Gemino. However, this issue can be mitigated by employing a one-time compression of the fixed key-value pairs derived from the reference frames or by utilizing faster techniques to approximate the attention mechanism [147, 125, 87].

1.3.3 Reparo: A Loss-Resilient Generative Codec

As described in §1.2.2, packet loss in a video conferencing application leads to corrupt or undecodable frames, which cause subsequent frames to be undecodable due to dependencies between frames. This causes an extended outage or glitch in a video call. Typically, this is resolved by retransmissions or FEC techniques, which are difficult to use for video conferencing applications under unreliable network conditions. Chapter 5 describes an alternative design in the form of a loss-resilient generative codec for video conferencing called Reparo. Reparo's design consists of two components. First, it represents all video frames using a small dictionary of visual tokens, where each token represents a fixed-size patch of a frame. The intuition behind this step is that most video frames can be captured by combining a limited set of patches. All subsequent operations happen on tokens. Specifically, the transmitter converts every frame into its corresponding token *indices* and transmits those in packets. When packets get lost in the network, the receiver in Reparo

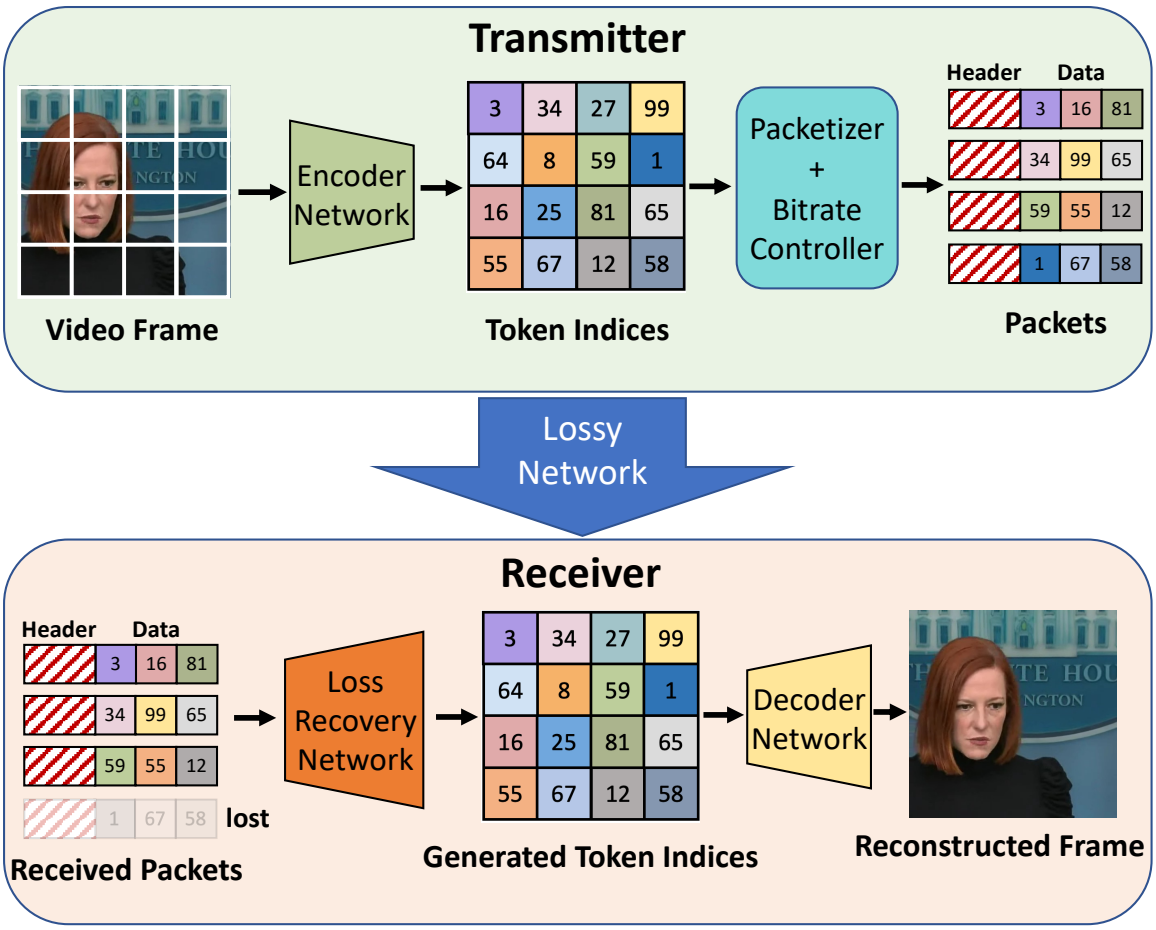


Figure 1-4: Overview of Reparo. An encoder-decoder pair converts between RGB image and quantized image tokens, while we introduce new modules to affect the packetization, bitrate, and loss recovery in the quantized token space to improve loss-resilience.

uses a neural network to infer any missing tokens based on correlations between the missing locations and the received tokens. These correlations are based on the neural network's conception of how tokens combine to create typical images in the visual world. Once the missing tokens are filled in, the image can be reconstructed by decoding the aggregate of the received and filled-in tokens. Fig. 1-4 illustrates the components of Reparo.

Beyond eliminating the need for transmitting redundant packets, Reparo offers three additional advantages. First, it achieves high compression rates without introducing dependencies between frames. Reparo exploits the natural dependencies within images in a video conferencing setting in creating its token dictionary or *codebook*. Consequently, Reparo does not need any temporal relationships across frames to achieve additional compression. This further means that only token indices (integers that are easily compressed) are transmitted

between the sender and the receiver as long as they have negotiated the codebook ahead of time.

Second, traditional codecs only achieve their target bitrate on average and often show a lot of fluctuation around an average because differences between adjacent frames could vary depending on motion on very short time-scales. This makes it very hard to design congestion control or error-recovery solutions such that network capacity is utilized well without latency spikes or packet loss. Since Reparo derives a fixed number of tokens from its target bitrate and only transmits token indices per-frame, it always matches its target bitrate. This predictability allows for many enhancements at the transport and application layer.

Third, Reparo requires only one-way communication. Existing FEC schemes [126] require an acknowledgment (ACK) for every frame successfully decoded to decide whether a retransmission is needed. This can cause longer freezes if the round-trip-time or RTT is high. In contrast, the Reparo receiver uses whatever tokens it receives to fill in the missing tokens on a continual and online basis without any explicit communication with the transmitter. 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 [17], and evaluate Reparo and VP9+Tambur [126], 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. However, since Reparo leverages attention and a transformer-based backbone, Reparo is more computationally intensive than traditional methods. Specifically, its current implementation requires multiple 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 leverage some of these techniques to improve Gemino, and anticipate that many similar techniques can be used to optimize Reparo too. We leave an exploration of such techniques to future work.

1.3.4 Beyond this Dissertation

We also make a number of other related contributions for video applications that are not included in the main body of this dissertation.

SEZMA [2]. The Single Endpoint Zoom Measurement Application (SEZMA) is a measurement tool to monitor user experience during video calls that can be run on individual's devices without cooperation from the other endpoint that they're conversing with. It captures application level statistics such as video quality and freezes, and correlates it with network level statistics such as packet sizes and packet loss. SEZMA, in contrast to existing measurement tools, does not rely on controlled experiments with contrived network environments, or need full control of all endpoints and routes between two or more conversing video calling users. Instead, SEZMA, logs network and video metrics in real-time while a user is on a video call, and sends them to a centralized server for post-processing. The application is designed to be lightweight, explanatory, usable, and privacy-preserving to improve users' likelihood of using it in the wild.

SRVC (ICCV 2021) [83]. SRVC is a content-adaptive super-resolution technique for on-demand video that achieves better compression without significant compute overheads. Similar to Gemino, SRVC leverages the compression benefits of existing video codecs at low resolution by transmitting downsampled video on the network. Specifically, SRVC, encodes video into two bitstreams: (1) a content stream, produced by compressing downsampled low-resolution video with an existing video codec, and (2) a model stream, which encodes periodic updates to a lightweight super-resolution network that is customized for short segments of the video. The key insight in SRVC is that by sending a selective set of updates to only those model parameters most relevant to the last few frames, the model stream can be compressed very efficiently. Similar to Gemino, SRVC first decodes the low-resolution video frames using the standard codec, and then upsamples and enhances it using its content-adapted model. Since SRVC is designed for on-demand video, the model and content streams can be pre-computed ahead of time to extract maximum compression gains.

Minerva (SIGCOMM 2019) [112]. Minerva is an end-to-end transport protocol for video streaming that achieves QoE fairness. Existing transport protocols only provide connection-level fairness without any heed to the video that the user is watching or the device the video is playing on. In contrast, Minerva uses information about the player state (buffer levels, video device, amount of motion in the video) to allocate bandwidth between video

users sharing a common bottleneck. Minerva clients do not explicitly communicate this state to each other, yet the server serving them converges to a bandwidth allocation between them that maximizes QoE fairness. In other words, Minerva optimizes the bandwidth allocation to automatically give a video client watching on a 4K display higher bandwidth than a client watching the same video on a mobile phone from the same household.

Vidaptive [77]. Vidaptive is a new transport design for real-time and low-latency video applications that achieves high network utilization without introducing latency spikes. Current video rate controllers like Google Congestion Control (GCC) [42] respond slowly to network changes, and have poor network utilization and latency profiles because their behavior is tightly coupled with the video encoder. Unfortunately, video encoders take time to match a target bitrate due to dependencies across frames, and also rarely achieve a consistent target bitrate even once caught up. To get around this without changing the video encoder, Vidaptive decouples packet transmission decisions from video encoder output, by introducing "dummy" padding traffic as needed to treat otherwise bursty video streams like backlogged flows controlled by a highly optimized congestion controller. Vidaptive then adapts the frame rate, resolution, and target bitrate of the encoder to align the video bitrate with the congestion controller's sending rate on very short time-scales. These two techniques make Vidaptive very responsive to any variations in network conditions while still maintaining high video quality and minimizing any frame latency spikes.

1.4 Previous Papers and Organization

Chapter 1 motivates the need for neural network-based techniques for video conferencing by describing challenging network conditions and the impact they have on the conferencing experience for end users. Chapter 2 describes the history of video conferencing and classical codecs, and also describes learning-based alternatives for video enhancement, synthesis and compression that have emerged in the last few years. The next three chapters and the main body of this dissertation are based on the following papers:

- Chapter 3 revises: Vibhaalakshmi Sivaraman, Pantea Karimi, Vedantha Venkatapathy, Mehrdad Khani, Sadjad Fouladi, Mohammad Alizadeh, Frédo Durand, Vivienne Sze. Gemino: Practical and Robust Neural Compression for Video Conferencing. In Proc. of USENIX NSDI 2024 [134].

- Chapter 4 revises: Vibhaalakshmi Sivaraman, Xuan Luo, Anne Menini, Mohammad Alizadeh, Rahul Garg. Multi-Resolution Multi-Reference Talking Head Synthesis via Implicit Warping. Technical Report [135].

- Chapter 5 motivates and revises: Tianhong Li, Vibhaalakshmi Sivaraman, Lijie Fan,

Mohammad Alizadeh, Dina Katabi. Reparo: Loss-Resilient Generative Codec for Video Conferencing. Reparo: Loss-Resilient Generative Codec for Video Conferencing. arXiv [92].

Chapter 6 acknowledges limitations of compression based on neural networks as well as limitations specific to Gemino and Reparo. Chapter 7 summarizes the key contributions of the thesis, and envisions a future where neural compression is more ubiquitous.

Chapter 2

Background and Related Work

2.1 History of Video Conferencing

Video conferencing history dates back to the late 80's and 90's when the first video codecs and compression techniques were developed. These compression techniques fundamentally moved the frontier enabling video storage and transmission in a way that was not possible previously. Specifically, the development of the Discrete Cosine Transform (DCT) [32] allowed images to be transformed into the frequency domain. The frequency domain made it easier to identify which features were most relevant to the eye (low-frequency) and which features could be thrown away (higher frequencies). During the 80s, multiple experiments aimed at extending DCT to the video case were conducted. This led to the the first video compression standards in the 90s with the development of MPEG by the Motion Pictures Experts Group [19, 11]. These standards introduced output formats for compressed video: macroblocks, motion vectors and DCT coefficients for the residuals. Though these codecs have evolved over the years and been optimized, the same output structures continue to be used even today.

Video conferencing necessitates a particular use case for video compression in a real-time setting. Frames read from web-cameras on a user's device are immediately compressed within the time bounds of the application, sent over the network, and decompressed and rendered at the receiver. These web-cameras, first introduced in the early 90s as well, used to sample every few seconds [21]. These sampling rates grew over time, eventually enough for smartphones to include applications like Skype, AOL Instant Messenger, and Yahoo Messenger with video calling functionalities in the 2000s. These capabilities further improved in the 2010s with the advent of the iPhone which supported video calling over cellular networks in addition to WiFi calling [18]. This steady growth in video conferencing was punctured when the COVID-19 pandemic induced a huge spike in usage in early 2020. Through 2020 and 2021, video applications like Zoom, Microsoft Teams, and Google

Meet saw a 21× growth in their monthly active users relative to their pre-pandemic levels [23]. This unprecedented growth has since plateaued; yet, the number of users of video conferencing remains high even as we transition to more in-person interactions. This is unsurprising – we all make frequent video calls with family living far from us, and many people still work in hybrid settings wherein they make video calls at least a few times a week.

Unlike the original web-cameras, today's web-cameras sample at 30 or even 60 frames per second to provide users good video calling experience. To support such high sampling rates and large user-bases, it is imperative that video codecs achieve high compression at high speeds. The rest of this chapter describes how current video codecs work, and what attempts have been made to design neural-network based alternatives for the same compression pipeline. After that, we describe video synthesis techniques more broadly since this dissertation repurposes many of those techniques for video compression.

2.2 Codec Design

2.2.1 Traditional Codecs

Most video applications rely on standard video compression modules (codecs) such as H.264/H.265 [129, 136], VP8/VP9 [34, 109], and AV1 [46]. These codecs separate video frames into keyframes (I-frames) that exploit spatial redundancies within a frame, and predicted frames (P-/B-frames) that exploit temporal—as well as spatial—redundancies across frames. Keyframes are compressed using standard image compression techniques using the DCT approach described above. After converting the spatial information to frequency domain using DCT, most of the low-frequency DCT coefficients are retained with high precision. The high-frequency DCT coefficients are encoded with lossy compression in a way that trades off the compression achieved with how distorted the decoded frame looks relative to the original because of missing high-frequency content. Predicted frames are further compressed based on their differences relative to a nearby keyframe or a previous predicted frame, using an approach called "delta encoding". Specifically, predicted frames are broken spatially into "macroblocks" that are square patches of varying size depending on the extent of detail in a particular location. Then, motion vectors are computed from a nearby frame to the current frame on a macroblock-by-macroblock basis that reflect which block from a keyframe can be repurposed where in the predicted frame. Any differences or "residuals" not captured by the motion vectors are then encoded using DCT and lossy compression. Delta encoding and the use of motion vectors makes video compression orders of magnitude more efficient than simple per-frame image compression. This is the only reason we can store videos and transmit them across the Internet.

Over the years, video codecs have been improved through ideas like variable block

sizes [136] and low-resolution encoding for lower bitrates [46]. These codecs are particularly efficient in their *slow* modes when they have generous time and compute budget to compress a video at high quality because they can perform extensive searches to compute the best motion vectors between macroblocks of the predicted frame and its nearby keyframe. However, these codecs still require a few hundred Kbps for real-time applications such as video conferencing, even at moderate resolutions like 720p. In low-bandwidth scenarios, these codecs cannot do much other than transmit at the worst quality, and suffer packet loss and frame corruption [54]. To circumvent this, some applications [8] switch to lower resolutions when the network degrades. However, as new video conferencing solutions such as Google's Starline [30] with a large bandwidth footprint are introduced, these concerns with current codecs become more acute.

2.2.2 Neural Codecs

The inability of traditional codecs to operate at extremely low bitrates for high-resolution videos has led researchers to consider solutions that use neural networks to reconstruct video from very compact representations. Neural codecs have been designed for video-streaming [83, 158, 49], live-video [84], and video video conferencing [149, 113]. Some of these models keep the fundamental structure of traditional codecs but replace it with learnt components. For instance, DVC [101] uses neural networks to perform motion estimation and encode residuals, while Swift [49] compresses and decompresses the residuals in a layered-encoding stack. Others leverage solutions to existing computer vision tasks for compression purposes. For instance, NAS [158] and LiveNAS [84] enhance video quality using one or more DNN models at either the client for video streaming, or the ingest server for live video. The models have knobs to control the compute overheads by using a smaller Deep Neural Network (DNN) [158], or by adjusting the number of epochs over which they are fine-tuned online [84]. We describe more of these vision tasks, and what advantages and disadvantages repurposing solutions for those tasks possess in a compression context in §2.3. Despite their differences, many of these approaches have shown improvements in the bits-per-pixel or bitrate consumption across a wide range of videos.

However, video conferencing differs from other video applications in a few ways. First, the video is unavailable ahead of time to optimize for the best compression-quality tradeoff. Moreover, the interactivity of the application demands that the video be both compressed and decompressed with low-latency. Second, the videos belong to a specific distribution consisting primarily of facial data. This allows for a more targeted model for generating videos of faces. A number of such models have been proposed [162, 149, 113, 132, 69, 143, 105] over the years. These models use keypoints or facial landmarks as a compact intermediary

representation of a specific pose, to compute the movement between two poses before generating the reconstruction. The models may use 3D keypoints [149], off-the-shelf keypoint detectors [162], or multiple reference frames [143] to enhance prediction. We describe these models and their shortcomings in more detail in §2.3. This dissertation proposes Gemino, a new codec that exploits the narrow distribution of facial data but uses a super-resolution approach to achieve higher fidelity synthesis than prior approaches.

2.2.3 FEC for Loss Recovery

Video codecs and standards govern how video is compressed to achieve a certain bitrate for storage as well as transmission purposes. However, when video is transmitted over the network, some of the encoded packets may be lost. One way to overcome this is to retransmit lost packets. However, this typically takes a few round-trip-times (RTTs) for the sender and the receiver to communicate. Forward Error Correction (FEC) is a technique commonly 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 [36], Reed-Solomon (RS) codes [120], and fountain codes [104] are all block codes that are optimal for random losses, where packets are lost independently. Recently, researchers have proposed using streaming codes for FEC [126], achieving better loss recovery capabilities than block codes for bursty losses, where several packets over one or more consecutive frames are lost. Bursty losses are more common in video conferencing settings because it is hard to control the encoder's output over short time-scales (§5.1). As a result, once the encoder's bitrate exceeds the available bandwidth, it is stuck in that state for many frames before it decreases its bitrate. This invariably causes packet loss. Since video frames are encoded with dependencies across frames, packets from a lost frame corrupt the decoder's state and cause an extended outage or frozen video over a few seconds. This dissertation describes Reparo, the first neural loss recovery scheme for video conferencing. Reparo 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 received data.

2.3 Learned Methods for Video Synthesis

This section describes different classes of vision tasks and tools that can be leveraged for video compression by transmitting a compact representation of video frames over the network and synthesizing the final video at the receiver from that compact representation.

First, we elaborate on view-synthesis techniques that take a reference frame (and pose) and generate a new target pose for the same person or identity from keypoints or audio. Then, we describe super-resolution techniques that upsample a low-resolution frame to produce its high-resolution counterpart. Lastly, we describe attention, a popular primitive in many generative models, and the capabilities it brings for video synthesis.

2.3.1 Novel-view synthesis

Face synthesis techniques fall under the broader problem of synthesizing a novel-view of an object or a person given a certain reference view. Many proposals [170, 115] have attempted to tackle the general version of the problem including Multi-View Image Fusion [139] and NeRF [107]. These approaches typically estimate a flow from the existing reference frame to the novel view. The flow captures which parts of the reference can be moved or copied over to the novel view, and consequently, what parts need to be synthesized because they are different from the reference.

Approaches specific to generating *faces* in a target pose based on reference or texture information have also been widely studied [131, 162, 149, 114, 144, 105, 134, 154]. For example, the First-Order Motion Model (FOMM) [131] animates a source image of a person into the target pose by estimating the motion using a first-order approximation around a set of ten sparse keypoints that are learnt end-to-end. Maxine [149] furthers this idea by using three-dimensional keypoints. The FOMM itself has been extended to use multiple source images [144] as well as optimized to run on mobile devices [114] albeit at much lower fidelity. Since keypoints provide a limited representation of the facial movement and mouth positioning, a few approaches [171, 166, 168, 121] leverage audio data in addition to pose information. All these approaches warp the source image based on the estimated optical flow between the source and target poses. However, such estimates can be incorrect when the source and target poses vary considerably. In this dissertation, we argue that using low-resolution frames instead of audio or keypoints provides more robust reconstructions with higher fidelity to the target frame.

2.3.2 Super-resolution

Super-resolution (SR) approaches take as input a low-resolution (LR) input and reconstruct a high-resolution (HR) version. The standard form within this realm is single-image SR techniques which only use the input image. Recently, CNN-based approaches [95, 98, 102, 85] that learn this transformation have significantly outperformed classic (non-learnt) interpolation methods such as bilinear or bicubic [81]. Since a number of these models tend to be large, efforts to distill SISR models into lightweight versions have resulted in newer models such as IDN [71] and IMDN [70]. Video SR methods [40, 93] build on image SR but further

improve the reconstruction by exploiting redundant information in adjacent low-resolution video frames. Certain approaches like FAST [165] and Nemo [157] further optimize SR for video generation by performing SR only on "anchor frames" and generating the rest by upsampling motion vectors and residuals. For video conferencing, domain-specific SR has also shown promising outcomes utilizing facial characteristics and training losses in their models [45, 103]. However, to the best of our knowledge, none of these prior methods study upsampling conditioned on a high-resolution image from the same context. Unlike pure SR methods, Gemino (Chapter 3) provides access to a high-resolution reference frame and learns models that jointly in-paint and propagate high-frequency details from the reference frame. In recent work, SRVC [83] uses content-specific super-resolution to upsample a low-resolution video stream. Our approach is similar to SRVC in that it designs a model adapted to a specific person. However, to enable real-time encoding, Gemino only customizes the model once per person rather than continuously adapting it throughout the video.

2.3.3 Attention-based Generative Techniques

Attention mechanisms, or techniques to focus on features or information within some fixed context window in a learnt pipeline, have gained popularity in vision tasks since the introduction of the self-attention layer (identical query, key, values) [142]. A number of models [167, 100, 37] for image and video-related tasks including ViT [51] and VCT [106] have since employed either self- or cross- attention. Since attention mechanisms are very compute and memory intensive, a number of recent efforts [48, 147, 87, 146] have either optimized or adapted the details of the attention layer to reduce its footprint.

These attention mechanisms have contributed to significant recent progress in the development of generative models, which can create text, audio, images, and videos that are indistinguishable from those created by humans [50, 99, 122, 91] often without conditioning inputs like low-resolution or similar images. 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 [50], and an image generative model can produce an image of an object using only a partial view [91]. Multi-modal generative models can even synthesize high-quality images from text prompts [123, 117]. Such techniques have been incorporated into certain super-resolution models [95, 155, 102] as well to help capture the long-range dependencies between the low-resolution input and high-resolution target better.

To enhance the use of domain knowledge, many recent visual generative models have adopted a two-stage design [141, 119, 43, 159, 89]. First, they learn to represent the target domain using a visual token dictionary or codebook. 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 massive performance improvements over earlier models in image generative tasks, such as text-to-image synthesis [122] and image editing [91] despite their use of discrete codes or quantized features. One such model applicable for video conferencing is Codeformers [169] which uses a codebook specific to faces, and uses attention to find correspondence between LR input patches and the corresponding HR patch from the codebook. However, it is a generative approach that may not synthesize images with the same identity as the HR ground-truth.

Given these capabilities, codebook- and token-based generative models are well-suited for loss-resilient video conferencing. Reparo [92] 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.

Chapter 3

Gemino: A Robust Low-Bitrate Neural Codec

3.1 Motivation

Neural synthesis approaches and specifically, keypoint-based models fall short in a number of ways that make them impractical in a video conferencing setting. These models operate similarly to the model described in Fig. 1-3 but do not transmit or use the downsampled target frame. They extract keypoints from the downsampled target frame, and transmit those instead. The receiver then reconstructs the target frame using its keypoints and a reference frame sent ahead of the video call. This choice of using keypoints causes major reconstruction failures when the reference and target frames are not close. Fig. 3-1 shows the reconstruction produced by the First-Order-Motion Model (FOMM) [132], a keypoint-based model, on 1024×1024 frames. We focus on the FOMM as a representative keypoint-based model, but these limitations extend to other such models. The FOMM only produces blurry outlines of the faces in rows 1 and 3 where the reference and target differ in orientation and zoom level respectively. In row 2, the FOMM misses the arm altogether because it was not present in the reference frame and warping alone cannot convey the arm's presence. Such failures occur because keypoints are limited in their representation of differences across frames, and most warping fields cannot be modeled as small deformations around each keypoint. Poor prediction quality in the event of such movements in video calls seriously disrupts the user experience.

Secondly, even in regions without much movement between the reference and the target frames, current approaches do not have good fidelity to high-frequency details. In row 2 of Fig. 3-1, the microphone does not move much between the reference and the target, but possesses a lot of high-frequency detail in its grille and stand. Yet, the FOMM has a poor reconstruction of that area. In row 1 of Fig. 3-1, it misses even those details in the hair that are similar between the reference and the target. This issue becomes more pronounced at higher resolutions where more high-frequency content is present in each

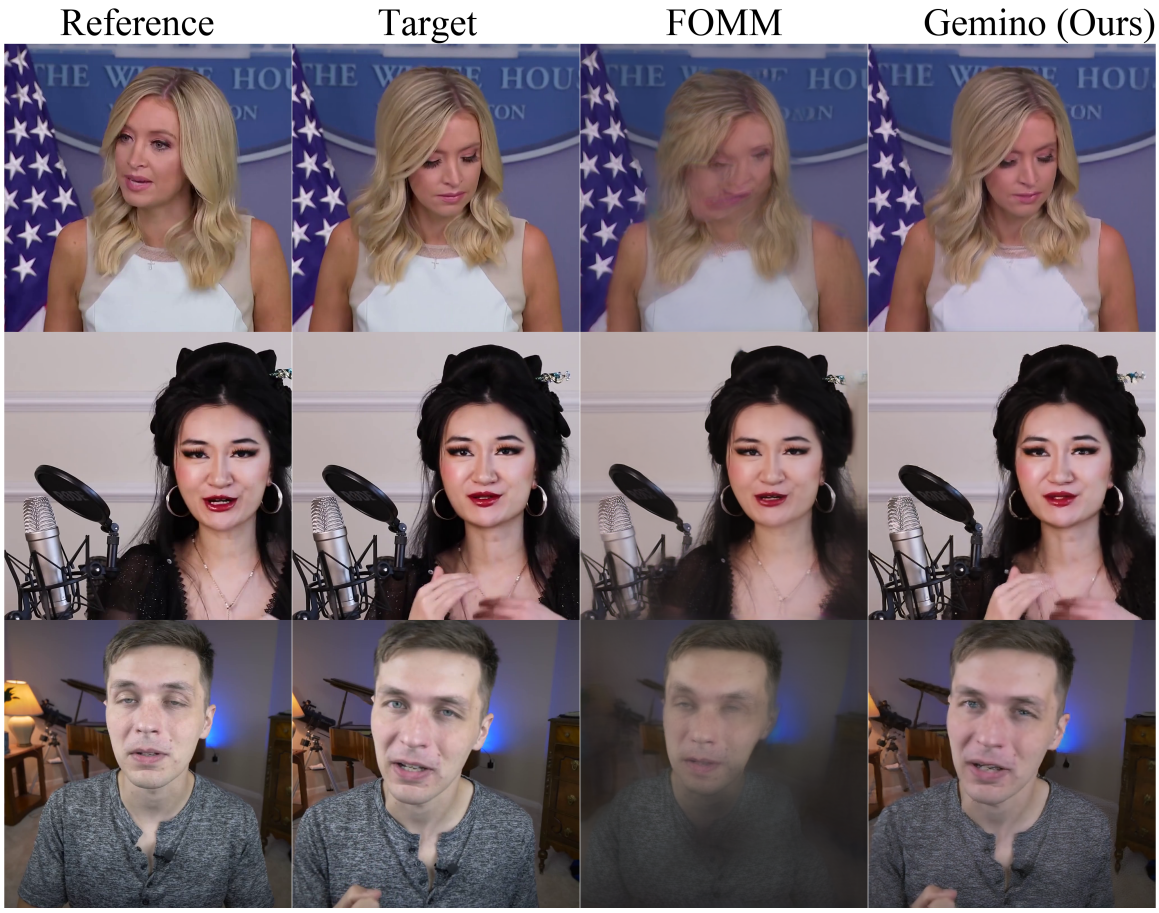


Figure 3-1: Failure cases in the FOMM's [132] reconstruction when the reference and target differ. FOMM misses the hand (row 2) because the reference frame does not contain it. In rows 1 and 3, FOMM only produces a blurry outline of the face (and torso). Gemino improves on these reconstructions by utilizing the downsampled target frame in its architecture and capturing the difference between the reference and the target better.

	First-Order-Motion Model			Gemino	Gemino (Opt.)
GPU	256×256	512×512	1024×1024	1024×1024	1024×1024
Titan X	17.35 ms	50.13 ms	187.30 ms	68.08ms	26.81ms
V100	12.48 ms	33.11 ms	117.27 ms	41.23ms	17.42ms
A100	8.19 ms	12.76 ms	32.96 ms	17.66ms	13.89ms

Table 3-1: Inference time of the First-Order-Motion Model [132] at different resolutions on different GPU systems. Prediction time increases with resolution, making it hard to reconstruct using the FOMM at 1024×1024 in real-time. The multi-scale architecture (third column) and further optimizations (fourth column) allow Gemino to achieve real-time inference at 1024×1024 on NVIDIA V100, A100, and Titan X systems.

frame, and where the human eye is more sensitive to missing details.

Lastly, many existing approaches [113, 132, 143] are evaluated on 256×256 images [111]. However, typical video conferences are at least 720p, especially in the full-speaker view. While the convolutions in these models allow them to scale to larger resolutions with the same kernels, the inference times often exceed the real-time deadline (33ms per frame for 30 frames-per-second) at high resolutions. We observe this in Tab. 3-1 which shows the inference time of the FOMM [132] on different GPU systems at different resolutions. This is unsurprising: running the FOMM unmodified at different resolutions performs the same number of operations per-pixel on 1024×1024 and 256×256 frames even though each pixel represents a smaller region of interest in the former case. With 16× more input pixels, the model's encoded features are also larger. This suggests that we need to redesign neural codecs to operate at higher resolutions without significant compute overheads, particularly as we move towards 4K videos.

3.2 System Design

3.2.1 Overview

Our goal is to design a robust neural compression system that reconstructs both low and high frequency content in the target frame with high fidelity even under extreme motion and variance across target frames. Ideally, such a system should also be able to operate at high resolution without significant compute overheads. Our prototype operates at 1024×1024 resolution.

Our key insight is that we need to go beyond the limited representation of movement that keypoints or facial landmarks alone provide. Fortunately, modern video codecs are very efficient in compressing low-resolution (LR) video frames. For instance, VPX compresses 128×128 resolution frames (8× downsampled in each dimension from our target resolution) using only 15 Kbps. Motivated by this observation and the fact that downsampled frames possess much more semantic information than keypoints, we design a model that performs *super-resolution conditioned on high-frequency textures* from a full-resolution reference frame. More specifically, our model resolves a downsampled frame at the receiver to its full resolution, aided by features derived from a full-resolution reference frame.

3.2.2 Model Architecture

Fig. 3-2 describes Gemino's model architecture running at the receiver of a video conferencing session. We assume that the receiver has access to a single *high-resolution reference frame*, capturing what the speaker looks like in that particular session. It also receives a stream of *low-resolution target frames* that are compressed by a traditional video codec. ① The receiver first takes the decompressed low-resolution (LR) target that it receives from

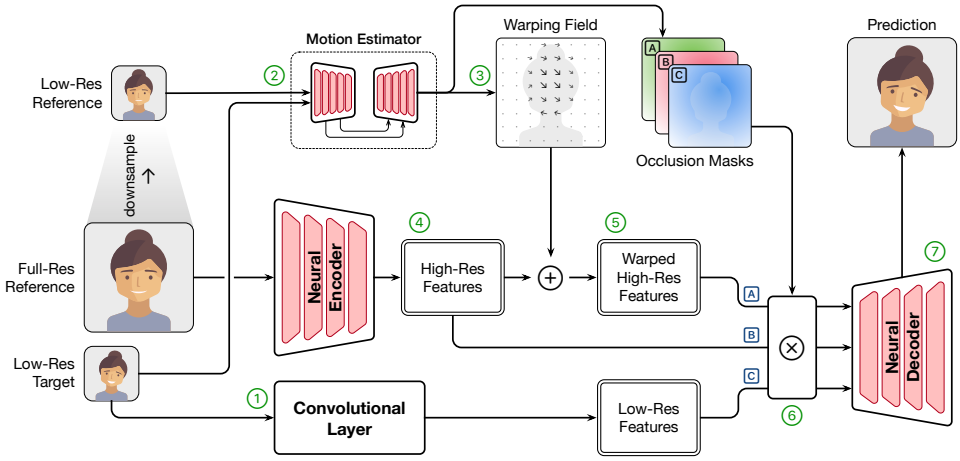


Figure 3-2: Gemino's high-frequency-conditional super-resolution model at the receiving client. The model first obtains features from the low-resolution target. It combines the low-resolution reference and target frames to produce a warping field based on the motion between them. The warping field is applied on encoded features from a full-res reference frame. The low-resolution and full-resolution features are jointly decoded by the neural decoder to produce the prediction at the receiver.

the sender over the network and runs it through a single convolutional layer to produce LR features. ② Next, the reference frame is downsampled and supplied, along with the (decompressed) LR target, to a motion estimation module that consists of a UNet [124]. ③ Since the two frames typically differ in their axes alignment, the motion estimator uses the keypoints to output a "warping field" or a transformation to move the reference frame features into the target frame's coordinates. ④ Meanwhile, the full-resolution reference frame is fed through a series of convolutional layers that encode it to extract high-resolution (HR) features. ⑤ The model applies the transformation produced by the motion estimator to the HR features to produce a set of warped (or rotated) HR features in the target frame's coordinate system.

Once the warped and encoded features from both the LR and HR pipelines are obtained, the model combines them before decoding the features to synthesize the target image. Specifically, ⑥ the model uses three inputs to produce the final reconstruction: Ⓐ the warped HR features, Ⓑ the HR features prior to warping, and Ⓒ the LR features. Each of these inputs on a feature-by-feature basis with three occlusion masks of the same dimension, produced by the motion estimator. These occlusion masks sum up to 1 and describe how to weigh different regions of the input features when reconstructing each region of target frame. For instance, mask Ⓐ maps to parts of the face or body that have moved in the reference and need to be regenerated in the target pose, mask Ⓑ maps to the regions that don't move between the reference and target (*e.g.,* background), while mask Ⓒ maps to

new regions (*e.g.,* hands) that are only visible in the LR frame. ⑦ Once multiplied with their respective masks, the combined input features are fed through convolutional layers in the decoding blocks that upsample them back into the target resolution. A more detailed description of the model with figures can be found in App. A.

Why is super-resolution insufficient? A natural question is whether the high-resolution information from the HR reference frame is essential or if we can simply perform super-resolution on the LR input frame through a series of upsampling blocks. While the downsampled LR frame is great at conveying low-frequency details from the target frame, it possesses little high-frequency information. This often manifests in the predicted frames as a blur or lack of detail in clothing, hair, or facial features such as teeth or eye details. To synthesize frames that are faithful to both the low and high-frequency content in the target, it is important to combine super-resolution from the LR frame with features extracted from the HR reference frame (§3.4.3). The former ensures that we good low-frequency fidelity, while the latter handles the high-frequency details. The last column of Fig. 3-1 shows how much the reconstruction improves with our high-frequency-conditional super-resolution approach. Specifically, we are able to capture the hand movement in row 2, the head movement in row 1, and the large motion in row 3. Gemino also produces a sharper reconstruction of facial features and the high-frequency content in the grille in row 2.

3.2.3 Optimizations To Improve Fidelity

Codec-in-the-loop training. Our design decision to transmit low-resolution frames instead of keypoints is motivated by modern codecs' ability [34, 109, 7] to efficiently compress videos at lower resolutions. Downsampled frames can be compressed at a wide range of bitrates depending on the resolution and quantization level. However, the LR video resolution determines the required upsampling factor. Note that different upsampling factors normally require different variants of the deep neural model (*e.g.,* different number of upsampling stages, different feature sizes, *etc.*). This means that for each target bitrate, we first need to choose a LR video resolution, and then train a model to reconstruct high-res frames from that resolution. We first create a reverse map from bitrate ranges to resolutions by profiling VPX codecs to identify the bitrate regime that can be achieved at each resolution. We use this map to select an appropriate resolution and codec for the LR video stream at each target bitrate. Tab. 3-2 shows the resolution and codec Gemino uses in each bitrate range. Once a resolution is chosen, we train the model to upsample *decompressed LR frames* (from the VPX codec) of that resolution to the desired output resolution. This results in separate models for each target bitrate regime, each optimized to learn the nature of frames (and artifacts) produced by the codec at that particular

Bitrate Regime	Resolution	Codec
<30 Kbps	128×128	VP8
30 Kbps – 75 Kbps	256×256	VP8
75 Kbps – 200 Kbps	512×512	VP9
>200 Kbps	1024×1024	VP9

Table 3-2: Mapping between desired bitrate regime and chosen resolution in Gemino.

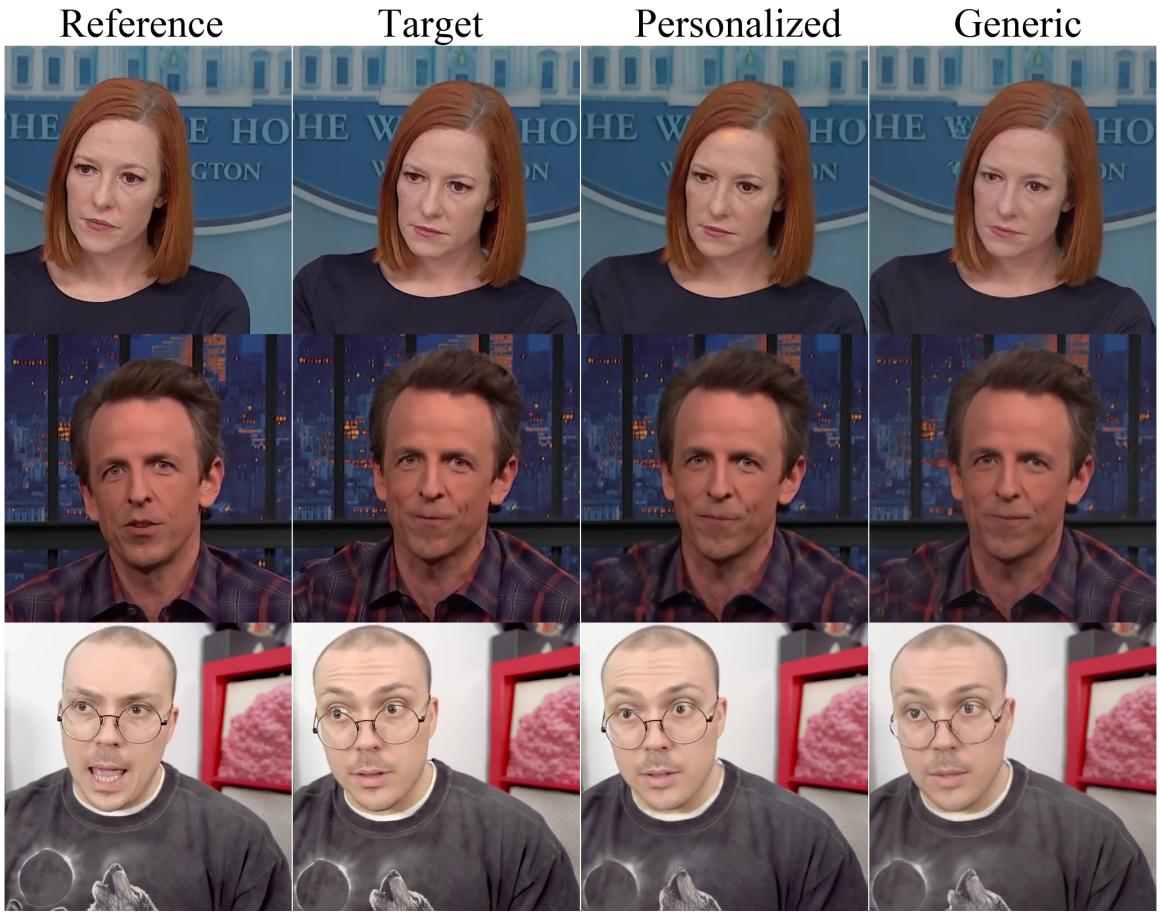


Figure 3-3: Performance of a model trained on a generic corpus of 512×512 videos compared to a personalized model fine-tuned on the specific person in the video. The personalized model better captures the eye gaze, dimples, and rim of the glasses.

resolution and bitrate. Since each such model requires 30 hours of training time on a single A100 GPU, we show that it is prudent to downsample to the highest resolution compressible at a particular bitrate, and use the model trained with a target bitrate at the lowest end of the achievable bitrate range for that resolution (§3.4.4).

Personalization. To improve the fidelity of our reconstructions, we explore training Gemino in two ways: *personalized* to each individual, and *generic*. To train the personalized model, we separate videos of each person into non-overlapping test and train data; the model

is not exposed to the test videos, but learns a person's facial features from training videos of that specific person. The generic model is trained on a corpus of videos consisting of many different people. Fig. 3-3 visually compares these two approaches. The personalized model is better at reconstructing specific high-frequency details of the person, e.g., eye gaze (row 1), dimples (row 2), and the rim of the glasses (row 3), compared to the generic model. We envision the model in neural video conferencing systems to be personalized to each individual using a few hours of their video calls, and cached at the systems of receivers who frequently converse with them. An unoptimized (full-precision) checkpoint of the model's 82M parameters is about a GB in size. However, we show that the model can be compressed in §3.4.4.

3.2.4 Reducing Computational Overheads

Multi-scale architecture. As we scale up the desired output resolution, it is crucial to perform fewer operations per pixel to reduce the compute overheads. We carefully examine different parts of the model in Fig. 3-2, and design a multi-scale architecture where we separate those modules that require fine-grained detail from those that only need coarse-grained information. Specifically, the motion estimation module is responsible for obtaining high-level motion information from the model. Even as the input resolution is increased, this coarse information can be inferred from low-resolution frames which retain low-frequency details. Consequently, in Gemino, we operate the motion estimation module on low-resolution reference and target frames (64×64). In contrast, the neural encoder and decoders are responsible for reconstructing high-frequency content and require fine-grained details from the high-resolution reference frame.

To further reduce computational overheads, we keep the bottleneck tensor at a fixed 64×64 resolution and simply feed in the LR features at the appropriate decoding layer based on the input resolution. For example, to produce 1024×1024 output from 128×128 input, the LR features are fed in like skip connections [124] after the first block at the expected 128×128 resolution. So as to not lose HR information through the bottleneck, we equip the first two blocks with skip connections [124] that directly provide encoded HR features to the decoder. As seen in the third column of Tab. 3-1, our multi-scale architecture reduces the reconstruction time of the model by 2.75× on older Titan X and NVIDIA V100 GPUs, while allowing us to easily meet the real-time deadline on an NVIDIA A100 GPU.

Model Optimizations. While the multi-scale architecture achieves real-time inference on A100 GPU system, it still fails to meet the 30 fps requirement on older systems such as Titan X. To enable this, we optimize our model further through a suite of techniques aimed at lowering the compute overheads of large neural networks. Specifically, we focus on reducing the number of operations or Multiply-Add Cumulations (MACs) involved in the expensive

decoding layers which are run on every frame. While a reduction in MACs does not always translate to reduction in latency on the targeted platform [156], it acts as a reasonable proxy for our case. We first replace the regular convolutional blocks in Gemino with depthwise separable convolutions (DSC) followed by a pointwise convolution [67, 153], a commonly used architecture for mobile inference. A standard convolution first filters all its inputs across all channels, then combine the results in one step resulting in a large number of matrix operations. In contrast, a depthwise convolution splits these actions into two separate layers: a depthwise separable layer that runs independent kernels across all channels and then combines them using a pointwise 1×1 convolution. The result is a $(1/n+1/k^2)$ factor decrease in the computational cost of a convolution with kernel size k with n output features.

To decrease the model MACs even further without impacting its fidelity, we run NetAdapt [156], a neural-architecture search algorithm that iteratively prunes the number of channels layer-by-layer until the compute overheads fall below a target threshold. NetAdapt can directly optimize for latency on different platforms, but we instead use it to optimize the MACs to reduce the overheads of running separate architecture searches for each platform. NetAdapt achieves its MAC reduction in iterations, each of which targets a small decrease (*e.g.,* 3%) towards its target. In each iteration, NetAdapt takes the current version of the model and generates a number of proposals for the next version of the model. Each proposal is generated by pruning a specific layer of the model until it meets the desired reduction in MACs. For every layer in the model, a target reduction in number of channels is first decided. After that, those channels whose weights' Frobenius norms are lowest are removed[1] from that layer. All the proposals are evaluated for their resulting accuracy after "short-term finetuning" on a small subset of the data. The proposal with the least accuracy loss is used as the starting model for the next iteration, and the pruning process repeats until the model has been shrunk sufficiently. Finally, the model is "long-term fine-tuned" on the entire dataset to recover its lost accuracy. Note that NetAdapt does not remove entire layers and only prunes channels within a layer. This might result in slightly suboptimal overall architectures.

Since Gemino uses personalized models, we adapt NetAdapt's logic to suit Gemino's training paradigm. We first apply NetAdapt on the generic Gemino model with short-term finetuning on the generic dataset. This is because neural architecture search is expensive to run, and we observe that the pruned architecture at the end of NetAdapt is the same for generic and personalized models. We finally long-term finetune the shrunk model in a personalized manner to better recover its accuracy (§3.4.4). These optimizations allow us

[1]With skip connections where a layer has outputs that are used in two places, we use the immediate following layer as opposed to the skip connection as the weights for which we calculate the Frobenius norm. For depthwise convolution, we look at the following pointwise convolution to decide which channels to remove, since that layer composes the outputs of the channels, as opposed to just filtering them.

to run inference on the Titan X GPU in 27 ms with barely any loss in visual quality (last column of Tab. 3-1) when upsampling 128×128 frames at 15 Kbps to 1024×1024. Given that video conferencing applications tolerate latencies of up to 200 ms (5–6 frames) in their jitter buffers [140], we believe that the additional delay from generating the received frame will be negligible.

3.2.5 Operational Flow

Training Procedure. To train Gemino, we first obtain weights from a trained FOMM model on the entire VoxCeleb dataset [111] at 256×256 resolution. We choose the appropriate training data for the specific person we want to train a Gemino model for, and train from scratch the additional downsampling and upsampling layers (that operate at higher resolutions) in the HR pipeline as well as all layers in the LR pipeline, while fine-tuning the rest of the model for 30 epochs. We repeat this procedure for different LR frame resolutions and target bitrate regimes mention in Tab. 3-2, and different people. In parallel, using the same procedure, we also train a generic version of the model on a larger corpus of people. Both models are replaced with depthwise-separable convolutions and optimized using NetAdapt [156] to produce the final model.

Inference Routine. Once versions of the model have been trained and optimized for different LR resolutions and target bitrates, we simply use the appropriate model for the current target bitrate regime and the person on the video call. The sender and the receiver pre-negotiate the reference frame at the beginning of the video call. This model performs inference on a frame-by-frame basis in real-time to synthesize the video stream at the receiver. We detail our prototype implementation and WebRTC pipeline further in §3.3.

3.3 Implementation

Basic WebRTC Pipeline. Our neural video conferencing solution uses WebRTC [152], an open-source framework that enables video and audio conferencing atop the real-time transport protocol (RTP) [128]. Since we perform neural frame synthesis, we use a Python implementation of WebRTC called *aiortc* [3] that allows easy interfacing with PyTorch. Aiortc handles the initial signaling and the peer-to-peer connection setup. A typical video call has two streams (video and audio) that are multiplexed onto a single connection. The sender extracts raw frames from the display, and compresses the video and audio components separately using standard codecs like VPX [34, 109], H.264/5 [129, 136], Opus [15], *etc.*. The receiver decompresses the received data in both streams before synchronizing them and displaying each frame to the client.

New Streams. We extend the standard WebRTC stack to use two distinct streams for

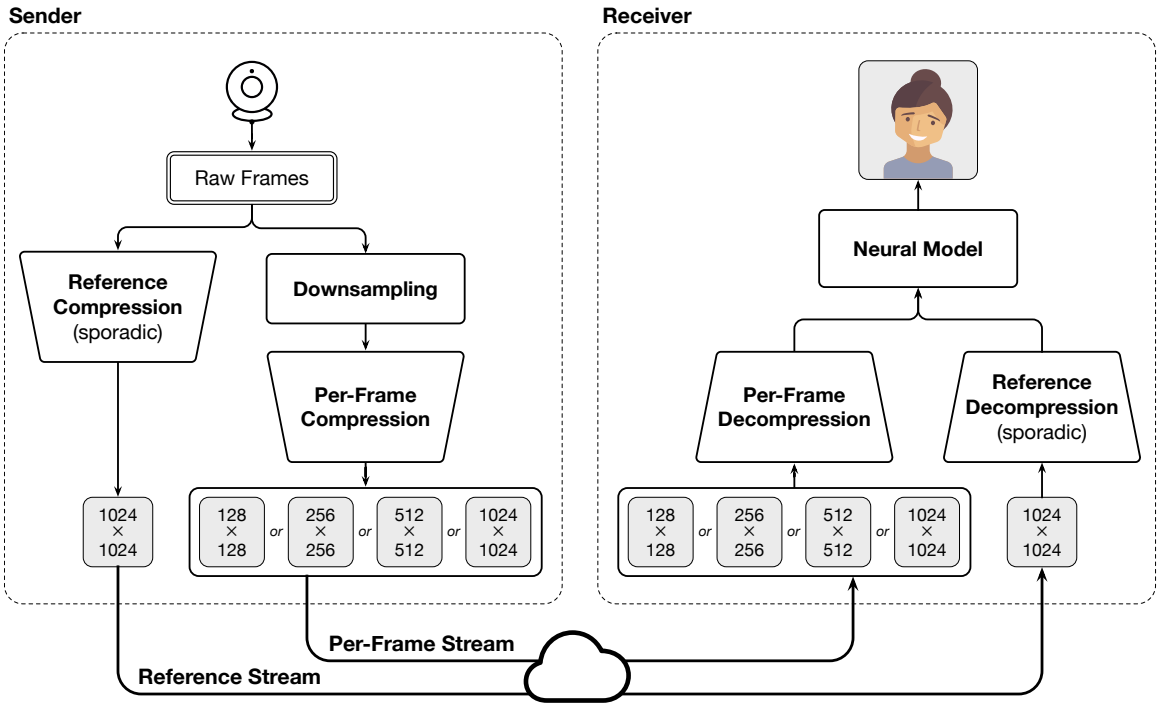


Figure 3-4: Neural video compression pipeline atop WebRTC [152]. We use two RTP streams: A sparse reference stream that sporadically sends high-resolution reference frames, and a per-frame (PF) stream that is used on every frame. The PF stream sends downsampled frames of the highest resolution that the current bandwidth can support, and thus has separate VP8 compression modules for each resolution. The receiver decompresses the downsampled frames, and supplies them, along with the latest reference frame, to the neural network that reconstructs the target video. If bandwidth is high enough, the PF stream is used for full-resolution VP8 frames without synthesis.

video: a *per-frame stream* (PF stream) that transmits downsampled video (e.g. 64×64 frames) *on every frame*, and a reference stream that transmits occasional but high-resolution *reference* frames that improve the synthesis fidelity. We anticipate using the reference stream extremely sparsely. For instance, in our implementation, we use the first frame of the video as the *only* reference frame. However, more reference frames may help recover high-frequency fidelity as it worsens when the reference and target frames drift apart. Most low-frequency changes between the reference and the target can be communicated simply through the downsampled target in the PF stream[2]. The receiver uses the per-frame information in the PF stream, with the reference information, to synthesize each high-resolution frame. Fig. 3-4 illustrates the expanded WebRTC architecture to accommodate

[2]We observe that sending reference frames with any fixed frequency adds significant bandwidth overheads. So, we only use a single reference frame in our evaluations. We leave an investigation of mechanisms to detect the need for a new reference frame (speaker moves significantly, high-frequency content or background changes) to future work.

the Gemino design.

The PF stream is implemented as a new RTP-enabled stream on the same peer connection between the sender and the receiver. We downsample each input frame to the desired resolution at the sender and compress it using the appropriate VPX codec. The frame is decompressed at the receiver. The bitrate achieved is controlled by supplying a target bitrate to VPX. Our PF stream can support full-resolution video that is typical in most video conferencing applications, while also supporting a range of lower resolutions for the model to upsample from. To enable this flexibility, we design the PF stream to have multiple VPX encoder-decoder pairs, one for each resolution that it operates at. When the sender transmits a frame, it chooses an appropriate resolution and codec based on the target bitrate, and compresses the video at that resolution and target bitrate. The resolution information is embedded in the payload of the RTP packet carrying the frame data. When the receiver receives each RTP packet, it infers the resolution and sends it to the VPX decoder for that resolution. Once decompressed, the low-resolution frame is upsampled by Gemino to the appropriate full-resolution frame. If the PF stream consists of 1024×1024 frames, Gemino falls back onto the regular codec and stops using the reference stream. The reference stream is repurposed from the existing video stream.

Model Wrapper. To enable neural frame synthesis, we define a wrapper that allows the *aiortc* pipeline to interface with the model. We reuse most of the pipeline from frame read at the sender to display at the receiver, except for introducing a downsampling module right after frame read, and a prediction function right before frame display. The wrapper is structured to perform format conversions and data movement from the AudioVisual [16] frames on the CPU that *aiortc* needs, to the PyTorch [116] tensors on the GPU required by the model. We initialize models separately for the sending and receiving clients. The wrapper also allows us to save (and periodically update) state at the sender and receiver which is useful for reducing the overheads from modules where we can reuse old computation (*e.g.,* run the encoder for high-resolution reference features only when the reference changes).

Further Optimizations. We optimize a number of other aspects of the *aiortc* pipeline. For instance, we move data between the CPU and GPU multiple times in each step of the pipeline. Batching these operations is difficult when maintaining low latency on each frame. However, to minimize PCIe overheads from repetitive data movement, we use *uint8* variables instead of *float*. We also keep reference frames and their encoded features stored as model state on the GPU. We pipeline as many operations as possible by running keypoint extraction, model reconstruction, and conversions between data formats in separate threads.

3.4 Evaluation

We evaluate Gemino in a simulation environment and atop a WebRTC-based implementation. We describe our setup in §3.4.1 and use it to compare existing baselines in §3.4.2. §3.4.3 motivates our model design, §3.4.4 discusses computational overheads and the impact of having the codec in our training process, and §3.4.5 shows that Gemino closely matches a time-varying target bitrate.

3.4.1 Setup

Dataset. Since most widely used datasets are of low-resolution videos [111, 47, 149] and lack diversity in the extent of the torso or face-zoom level, we collected our own dataset comprising of videos of five YouTubers with publicly available HD (1920×1080) videos. For each YouTuber, we curate a set of 20 distinct videos or URLs that differ in clothing, hairstyle, accessories, or background. The 20 videos of each YouTuber are separated into 15 training videos and 5 test videos. For each video, we manually record and trim the segments that consist of talking individuals; we ignore parts that pan to news segments or different clips. The segments are further split into 10s chunks to generate easily loadable videos for training, while the segments of the test video are combined to form a longer video. We also spatially crop each frame into our desired dimensions (typically 1024× 1024). based on the average location of the person across all frames of the video. Note that the number of pixels in a 1024×1024 frame is comparable to a 720p frame. We strip the audio since our focus is on video synthesis. Tab. A-1 in App. A.3 describes the details of the dataset. We use the 512×512 dataset from NVIDIA [149] to train a generic model to illustrate the benefits of personalization. Our evaluation focuses on reconstructing a single front-facing person in a video call; Gemino can be extended to multiple speakers if there are application-level techniques to separate speakers into individual streams [10, 9].

Model Details. The main model we evaluate is our high-frequency conditional super-resolution model that consists of an upsampling module that takes in features from a low-resolution (LR) frame, and upsamples it to 1024×1024. To provide the high-frequency details, it uses two pathways consisting of warped and unwarped features from the high-resolution (HR) reference image (Fig. 3-2). We use the first frame of the video as the *sole reference* image for the entire test duration. The warping field is produced by a motion estimation network that uses the first-order approximation near 10 keypoints [132]. Our multi-scale architecture runs motion estimation always at 64×64 irrespective of the input video resolution. The neural encoder (for the HR features) and decoder (for both LR and HR features) consist of four down and upsample blocks. The discriminator operates at multiple scales and uses spectral normalization [108] for stability. Layers of our model that

are identical in dimensions to those from the FOMM are initialized from a public FOMM checkpoint trained on the VoxCeleb dataset [111], and fine-tuned on a per-person basis. The remaining layers are randomly initialized and trained on a per-person basis over 30 epochs. We fine-tune the FOMM baseline also in the same personalized manner. We use Adam optimizer [86] to update the model weights with a learning rate of 0.0002, and first and second momentum decay rates of 0.5 and 0.999. We use equally weighted multi-scale VGG perceptual loss [75], a feature-matching loss [148], and a pixel-wise loss. We also use an adversarial loss [59] with one-tenth the weight of remaining losses. The keypoints use an equivariance loss similar to the FOMM [132]. We train our models to reconstruct from decompressed VPX frames corresponding to the low-resolution target frame so that the model learns to correct any artifacts produced by VPX.

Evaluation Infrastructure. We evaluate our neural compression system in a simulation environment where frames are read from a video, downsampled (if needed) for the low-resolution PF stream, compressed using VPX's chromium codec [24], and passed to the model (or other baselines) to synthesize the target frame. Note that the FOMM [132] uses keypoints and four "Jacobian" values around each keypoint for producing its warping, and transmits them over the network. We design a new codec for the keypoint data that achieves nearly lossless compression and a bitrate of about 30 Kbps. For VPX, we compress and decompress the full-resolution frame at different target bitrates, and measure the difference in visual quality between the output and the original frame.

To obtain end-to-end latency measurements and to demonstrate Gemino's adaptability to different target bitrates, (§3.4.5), we use our *aiortc* [3] implementation. A sending process reads video from a file frame-by-frame and transmits it to a receiving process that records each received frame. The two processes, running on the same server, use the ICE signaling [25] mechanism to establish a peer-to-peer connection over a UNIX socket, which then supports video frame transmission using the Real-Transport Protocol (RTP). We timestamp each frame as it is sent and received, and save the sent and received frames in their uncompressed forms to compute latency and visual metrics. We also log RTP packet sizes to compute the bitrate.

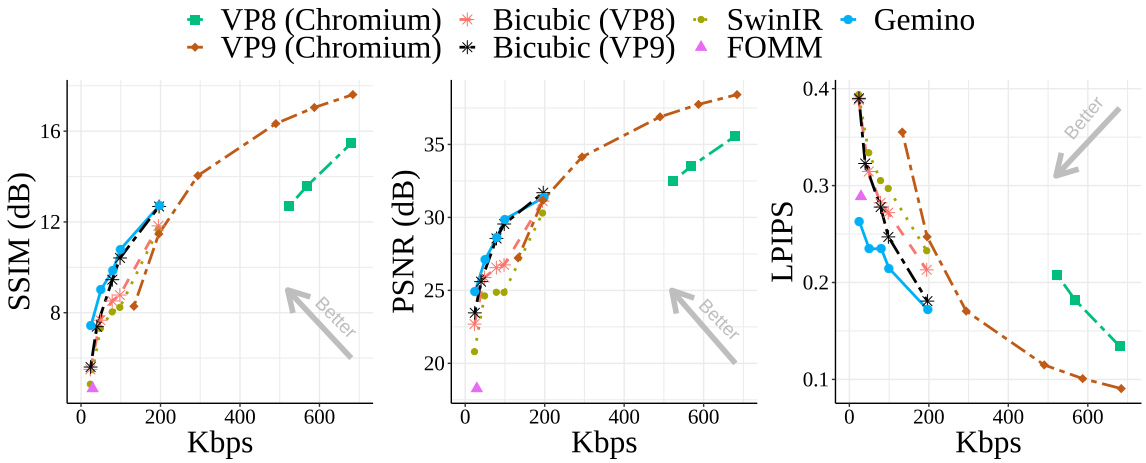
Metrics. To quantify the esthetics of the generated video, we use standard visual metrics such as PSNR (peak signal-to-noise ratio), SSIM (structural similarity index) in decibels [151], and LPIPS (learned perceptual image patch similarity) [163]. For PSNR and SSIM, higher is better; while for LPIPS, lower is better. We observe that differences in LPIPS are more reflective of how natural the synthesized frame feels and use that as our main comparison metric (§B.2); we also show visual strips where appropriate. We report the bitrate consumed to achieve a particular visual quality by measuring the total

data transferred (size of compressed frames or RTP packet sizes) over the duration of the video, and dividing it by the duration itself. To measure the end-to-end latency, we record the time at which the frame is read from the disk at the sender as well as the time at which prediction completes at the receiver. We report the difference between these two timestamps as our per-frame latency metric. We also report the inference time per frame when running the trained model in simulation; this does not capture the overheads of data conversion or movement in an end-to-end pipeline. This inference time needs to be $<$33ms to maintain a 30fps video call. We run all our experiments for the entire duration of each test video in our dataset (Tab. A-1), and report the average over all frames for each metric.

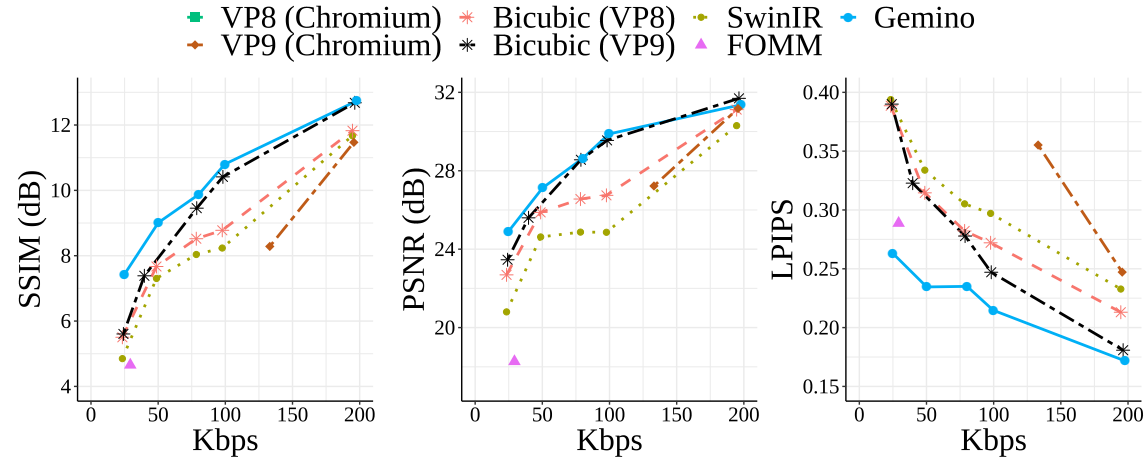
Baselines. We obtain the bitrate for *VP8*, the default codec in its Chromium settings [24] that comes with the *aiortc* codebase. We also implement and evaluate VP9 in the same setup. To evaluate the benefits of using a neural approach to video conferencing, particularly at lower bitrates, we compare a few different models: (1) *FOMM* [132], a keypoint-based model for face animation, (2) our approach, Gemino, (3) state-of-the-art super-resolution model based on *SwinIR* [94], and (4) *bicubic upsampling* [81] applied to the low-resolution VPX target frame. All of the compared models generate 1024$\times$ 1024 frames except for the generic model that uses NVIDIA's 512$\times$512 corpus [149].

3.4.2 Overall Bitrate vs. Quality Tradeoff

To quantify the improvements of our neural compression system, we first compare Gemino with VP8 and VP9 in their chromium configuration [24]. Fig. 3-5 shows the rate-distortion curve for all schemes. For VPX, we alter the target bitrate alone for full-resolution (1024$\times$1024) frames in the PF stream. For Gemino, bicubic, and SwinIR, we vary the resolution and target bitrate of the low-resolution (LR) frame in the per-frame (PF) stream. For each point on the rate-distortion curve for Gemino, we train a personalized model to reconstruct full-resolution frames from LR frames encoded at the highest resolution supported by that target bitrate. We motivate using the codec in training and choosing the resolution in §3.4.4. The PF stream is compressed with VPX's Chromium settings at the target bitrate for its resolution. We configure Gemino to choose the base codec (VP8 or VP9) that has the better visual quality for a given target bitrate. We plot the resulting bitrate and visual metrics averaged across all 25 test videos' (5 speakers; 5 videos each) frames. Fig. 3-5(a) shows that VP8 operates in a different bitrate regime than all other schemes. VP9 improves the dynamic range to about 130 Kbps, but still can't compress below that. Both VPX codecs operate on full-resolution frames, which cannot be compressed as efficiently as LR frames by the codec in its real-time mode. We observe that VP8 and VP9 consume 523 Kbps and 230 Kbps respectively to achieve an LPIPS



(a) Overall rate-distortion curve for all schemes



(b) Rate-distortion curve in low-bitrate regimes.

Figure 3-5: Rate-distortion curve for Gemino compared with existing baselines. VP8 and VP9 require ∼5× and ∼3× the bitrate consumed by Gemino to achieve comparable LPIPS. At lower bitrates, Gemino outperforms other approaches that upsample low-resolution video frames. Gemino's benefits become prominent as the bitrate regime is lowered.

of 0.21. In contrast, our approach is able to achieve the same visual quality with only 105 Kbps, providing nearly 5× and 2.2× in improvement over VP8 and VP9 respectively.

However, since our goal is to enable video conferencing in bitrate regimes where current codecs fail, we focus on bitrates < 200 Kbps in Fig. 3-5(b). Gemino uses VP8 at 15 Kbps and 45 Kbps but VP9 at higher bitrates; VP8 and VP9 do not differ much at lower bitrates, but VP9 allows even 512×512 to be compressed to 75 Kbps. Fig. 3-5(b) suggests that while LPIPS always shows considerable variation across schemes, small differences (1–2 dB) in PSNR and SSIM start manifesting only at lower bitrates. Specifically, Gemino

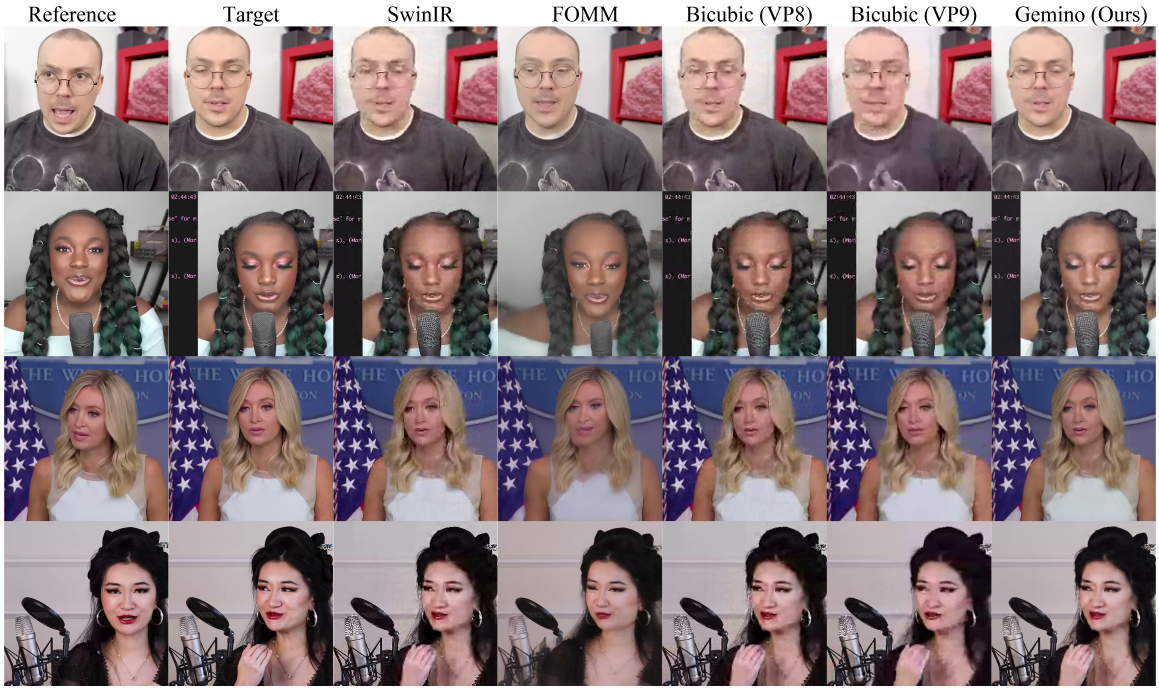


Figure 3-6: Visual comparison across low-bitrate baselines. All but FOMM upsample a 256×256 frame at 45 Kbps. Gemino's reconstructions contain a smoother output than the blocky artifacts observed with Bicubic and SwinIR. The FOMM completely fails when the reference and the target differ considerably in rows 1 and 2.

achieves over 1 dB better SSIM and PSNR, and 0.08 lesser LPIPS than Bicubic around 50 Kbps. To map these metrics to improvements in visual quality, we show some snippets in Fig. 3-6 and enlarged frames (with per-frame metrics) in §B.2. Fig. 3-6 shows that compared to Gemino, reconstructions from bicubic have more block-based artifacts in the face, and the FOMM misses parts of the frame (row 4) or distorts the face. SwinIR, a super-resolution model, performs worse than bicubic. We suspect this is because SwinIR is not specifically trained on faces, and is also oblivious to artifacts from the codec that it encounters in our video conferencing pipeline. To account for this, we evaluate a simple upsampling model personalized on faces in §3.4.3.

We also show CDFs of the visual quality in LPIPS across all frames in our corpus in Fig. 3-7 at 15 Kbps, 45 Kbps, 75 Kbps and 105 Kbps. The same CDFs for PSNR and SSIM can be found in §B.1. At each target bitrate, we use the largest resolution supported by the underlying video codec. The CDFs show that Gemino's reconstructions are robust to variations across frames and orientation changes over the course of a video. Gemino outperforms all other baselines across all frames and metrics. Specifically, its synthesized frames at 45 Kbps are better than FOMM by 0.05–0.1. It also outperforms Bicubic and SwinIR by 0.05 and 0.1 in LPIPS at the median and tail respectively. As we move from

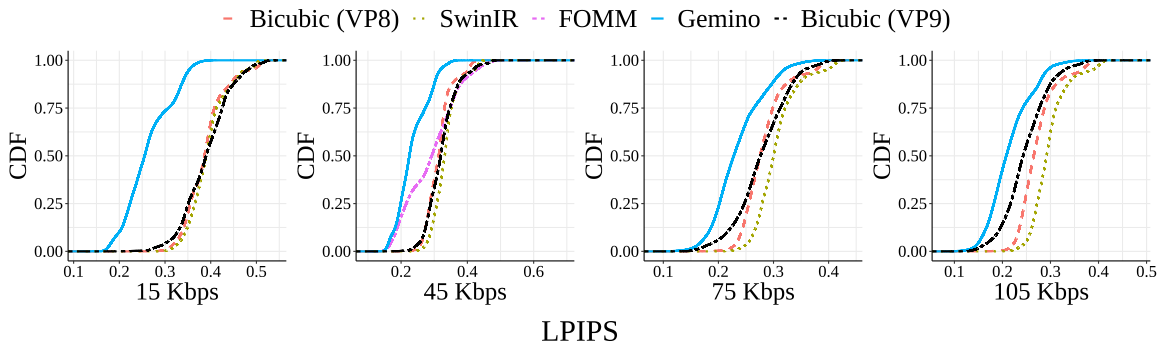


Figure 3-7: CDF of reconstruction quality across all video frames as that shows that, as we move from higher bitrates to lower, the improvement from Gemino relative to Bicubic, particularly over VP9, becomes more pronounced.

Model Architecture	PSNR (dB)	SSIM (dB)	LPIPS
Upsampling	24.72	7.02	0.30
Gemino	**24.90**	**7.42**	**0.26**

Table 3-3: Average visual quality of synthesized frames from Gemino and "Pure Upsampling" when reconstructing from decompressed 128×128 frames.

higher bitrates to lower, we observe that the improvement from Gemino relative to Bicubic, particularly over VP9, becomes more pronounced. While the gap in LPIPS at the median between Bicubic (VP9) and Gemino is less than 0.05 at 105 Kbps, it increases to nearly 0.2 at 15 Kbps. Gemino increases the end-to-end frame latency to ∼87 ms, from VPX's 47 ms. However, our carefully pipelined operations are optimized to enable a throughput of 30 fps.[3]

3.4.3 Model Design

Architectural Elements. To understand the impact of our model design, we compare Gemino to a *Pure Upsampling* approach from the LR target features with no pathways from an HR reference frame (only Ⓒ in Fig. 3-2) at decompressing VP8 128×128 frames at 15 Kbps to its full 1024×1024 resolution. Tab. 3-3 reports the average visual quality while Fig. 3-8 shows a CDF of performance across all frames and videos. Gemino outperforms Pure Upsampling on average by 0.4 dB on SSIM and 0.04 on LPIPS. This is also visible in rows 2 and 3 of Fig. 3-9 where the Pure Upsampling approach misses the high-frequency details of the microphone grille and lends blocky artifacts to the face respectively. This manifests as a 0.05 difference in the LPIPS of the median frame reconstruction and a nearly 0.1 difference between Gemino and Pure Upsampling at the tail in Fig. 3-8. Since one

[3]We ignore the delay due to the expensive conversion between NumPy arrays and Python's Audio Visual Frames in calculating latency because the the current PyAV library [16] is not optimized for conversion at higher resolutions.

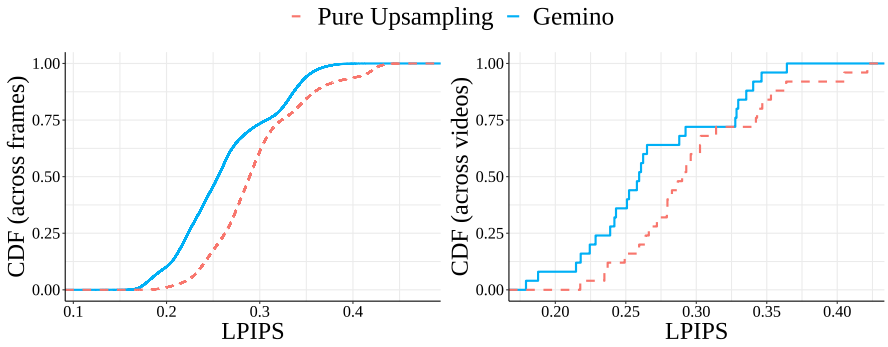


Figure 3-8: CDF of reconstruction quality of different model architectures across frames and videos in our test corpus. Gemino outperforms Pure Upsampling by nearly 0.05 in LPIPS at the median of both frames and videos. It also outperforms the approach that relies on only warped HR, and the RGB-based warping across most frames in the corpus.

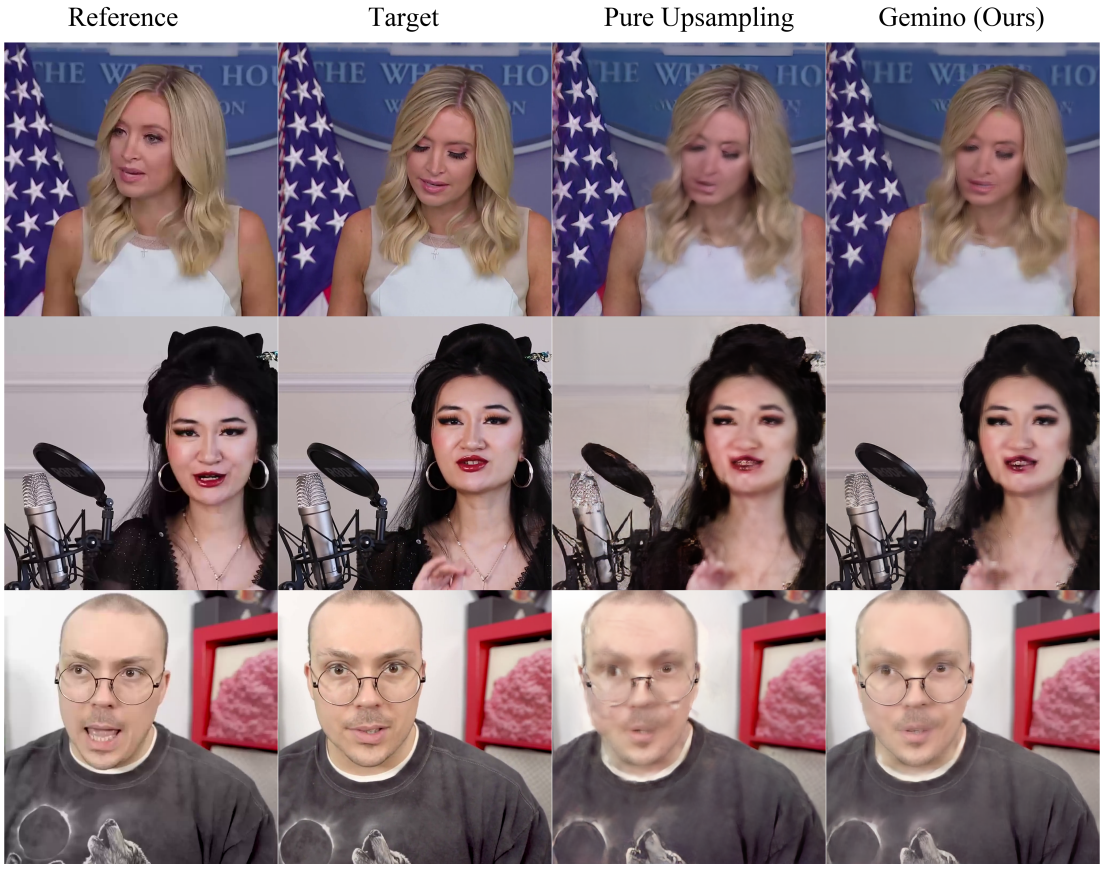


Figure 3-9: Visual comparison across different model architectures. Compared to Gemino, Pure upsampling misses the high-frequency details of the microphone (row 2) and lends block-based artifacts on the face (row 3).

of our goals is to have a robust neural compression system that has good reconstruction quality across a wide variety of frames, this difference at the tail across approaches is

Model	Training Regime	PSNR (dB)	SSIM (dB)	LPIPS
Gemino	Personalized	**30.59**	**11.11**	**0.13**
	Generic	29.48	10.55	0.13
Upsampling	Personalized	**30.32**	**11.10**	**0.13**
	Generic	28.89	9.59	0.17

Table 3-4: Impact of personalization on different models.

particularly salient. On the other hand, Gemino synthesizes the microphone better, and has fewer blocky artifacts on the face.

Personalization. To quantify the impact of personalization, we compare two versions of Gemino and Pure Upsampling: a *generic model* trained on a large corpus of 512×512 videos [149] of different people and a personalized model fine-tuned on videos of only a specific person. We compare the visual metrics averaged across all videos of all people when synthesized from a single generic model against the case when each person's videos are synthesized from their specific model. All models upsample 64×64 LR frames (without codec compression) to 512×512. Fig. 3-3 visually shows that the personalized Gemino model reconstructs the details of the face (row 1), dimples (row 2) and rim of the glasses (row 3) better but Tab. 3-4 shows that these personalization benefits are limited for Gemino (PSNR and SSIM improvements of 1.1 dB and 0.56 dB). However, personalization provides more pronounced benefits of nearly 1.5 dB in SSIM and PSNR, and 0.04 in LPIPS for the Pure Upsampling architecture. This suggests that the benefits from personalization are most notable when the underlying model architecture itself is limited in its representational power: the personalized Pure Upsampling architecture is able to encode high-frequency information in its weights when trained in a personalized manner, while it has no pathways to obtain that information in a generic model. It also illustrates that Gemino is more robust; it operates better than Pure Upsampling in the harder settings of a generic model and extreme upsampling from a 128×128 frame compressed to 15 Kbps (Fig. 3-8).

3.4.4 Operational Considerations

Computational Overheads. Tab. 3-5 explores the accuracy *vs.* compute tradeoffs of different models at 512×512 resolution with and without personalization as measured on different GPU systems with the various optimizations described in §3.2.4. We focus on optimizing the decoding layers (bottleneck and up-sampling) of the generator (Fig. A-3) since they are the most compute intensive layers, and are also run on every received frame. The accuracy is reported in the form of the average LPIPS [163] across all frames of our corpus. We observe that DSC reduces the decoder to 11% of its original MACs. While this

Convolution Type	**Regular**			**Depthwise**		
NetAdapt Target	None	10%	1.5%	None	10%	1.5%
# of Decoder MACs	195B	14.7B	2.4B	22.7B	1.7B	0.3B
# of Decoder Parameters	30M	1.3M	151K	3.4M	186K	26K
A100 Inference	13ms	9ms	8ms	11ms	8ms	7ms
V100 Inference	14ms	11ms	8ms	14ms	8ms	6ms
Titan X Inference	46ms	17ms	12ms	53ms	26ms	13ms
Jetson TX2 Inference	800ms	274ms	165ms	436ms	183ms	87ms
LPIPS (Generic)	0.13	0.15	0.17	0.14	0.16	0.19
LPIPS (Personalized)	0.13	0.13	0.15	0.14	0.14	0.17

Table 3-5: Accuracy and compute overheads of different versions of our model operating on 512×512 resolution.

PF Stream Resolution	PSNR (dB)	SSIM (dB)	LPIPS
64×64	23.80	6.77	0.27
128×128	25.72	7.86	0.27
256×256	**27.12**	**9.01**	**0.24**

Table 3-6: Reconstruction quality from different resolution PF stream frames at the same bitrate of 45 Kbps. Gemino reconstructs better from higher resolution frames.

gives limited improvements on large GPU systems, it improves the inference time on Jetson TX2, an embedded AI device, by 1.84× with a slight decrease in visual quality. Running NetAdapt further reduces the inference time to 87 ms at 1.5% of the model MACs on the TX2. The NVIDIA compiler on the Titan X and Jetson TX2 GPU systems is not optimized for DSC [153]; this can be improved with a TVM compiler stack [44] and optimized engines such as TensorRT [14]. However, running NetAdapt produces a real-time model for Titan X even at 10% of the original model MACs. As expected though, there is a loss of accuracy as the models become smaller. This loss is negligible in moving from the full model MACs to 10%, when personalizing (no change in LPIPS), but is more significant at 1.5% (0.02–0.03 increase in LPIPS). The generic model takes a bigger hit to its accuracy as it is pruned; its LPIPS increases by 0.02 with 10% pruning, and by 0.04 when pruned to 1.5% of the original model. The trend with personalization is expected since smaller models do not generalize well with their limited capacity, however even such fine-tuning does not help if the optimizations are extreme. This illustrates that there is a sweet spot (such as decreasing MACs to 10%) wherein the gains from decreased compute outweigh the loss (or lack thereof) in accuracy.

Choosing PF Stream Resolution. Gemino is designed flexibly to work with LR frames of any size (64×64, 128×128, 256×256, 512×512) to resolve them to 1024×1024 frames,

and to fall back to VPX at full resolution if it can be supported. VP8 and VP9 achieve different bitrate ranges at every resolution by varying how the video is quantized. For instance, on our corpus, we observe that 256×256 frames can be compressed with VP8 in the 45 Kbps–180 Kbps range, but VP9 can compress even 512×512 frames from 75 Kbps onwards. These bitrate ranges often overlap partially across resolutions. This begs the question: given a target bitrate, what resolution and codec should the model use to achieve the best quality? To answer this, we compare the synthesis quality with Gemino atop VP8 from three PF resolutions, all at 45 Kbps in Tab. 3-6. Upsampling 256×256 frames, even though they have been compressed more to achieve the same bitrate, gives a nearly 4 dB improvement in PSNR, more than 2 dB improvement in SSIM, and a 0.03 improvement in LPIPS, over upsampling lower resolution frames. This is because the extent of super-resolution that the model performs decreases dramatically at higher starting resolutions. This suggests that for any given bitrate budget, we should start with the highest resolution frames that the PF stream supports at that bitrate, even at the cost of more quantization or compression to achieve that bitrate. This also means that if VP9 can compress higher resolution frames than VP8 at the same target bitrate, we should pick VP9. Tab. 3-2 shows the resolution and codec we choose for different target bitrate ranges in our implementation.

Encoding Video During Training. A key insight in the design of Gemino is that we need to design the neural compression pipeline to co-exist with traditional codecs as well as leverage the latest developments in codec design. One way to do so is to allow the model to see decompressed frames at the chosen bitrate and PF resolution during the training process so that it learns the artifacts produced by the codec. This allows us to get extremely low bitrates for LR frames (which often causes color shifts or other artifacts) while maintaining good visual quality. To evaluate the benefit of this approach, we compare five training regimes for Gemino when upsampling 128×128 video to 1024×1024: (1) no codec, (2) VP8 frames at 15 Kbps, (3) VP8 frames at 45 Kbps, (4) VP8 frames at 75 Kbps, (5) VP8 frames at a bitrate uniformly sampled from 15 Kbps to 75 Kbps. We evaluate all five models at upsampling decompressed frames at 15 Kbps, 45 Kbps, and 75 Kbps.

Tab. 3-7 shows the LPIPS achieved by all the models in each reconstruction regime. All models trained on decompressed frames perform better than the model trained without the codec. Further, the model trained at the lowest bitrate (15 Kbps) performs the best even when provided decompressed frames at a higher bitrate at test time because it has learned the most challenging upsampling task from the worst LR frames, and performs well even with easier instances or higher bitrate frames. This suggests that we only need to train one personalized model per PF resolution at the lowest bitrate supported by a resolution, and then we can reuse it across the entire bitrate range that the PF resolution can support.

Training Regime	PF @ 15 Kbps	PF @45 Kbps	PF @75 Kbps
No Codec	0.32	0.30	0.28
VP8 @ 15 Kbps	**0.26**	**0.25**	**0.23**
VP8 @ 45 Kbps	0.28	0.27	0.25
VP8 @ 75 Kbps	0.30	0.28	0.26
VP8 @ [15, 75] Kbps	0.28	0.26	0.25

Table 3-7: LPIPS for different regimes wherein we include the VP8 codec in the training pipeline. The model trained with the lowest bitrate videos at a given resolution performs best regardless of what the bitrate of the video is at inference time.

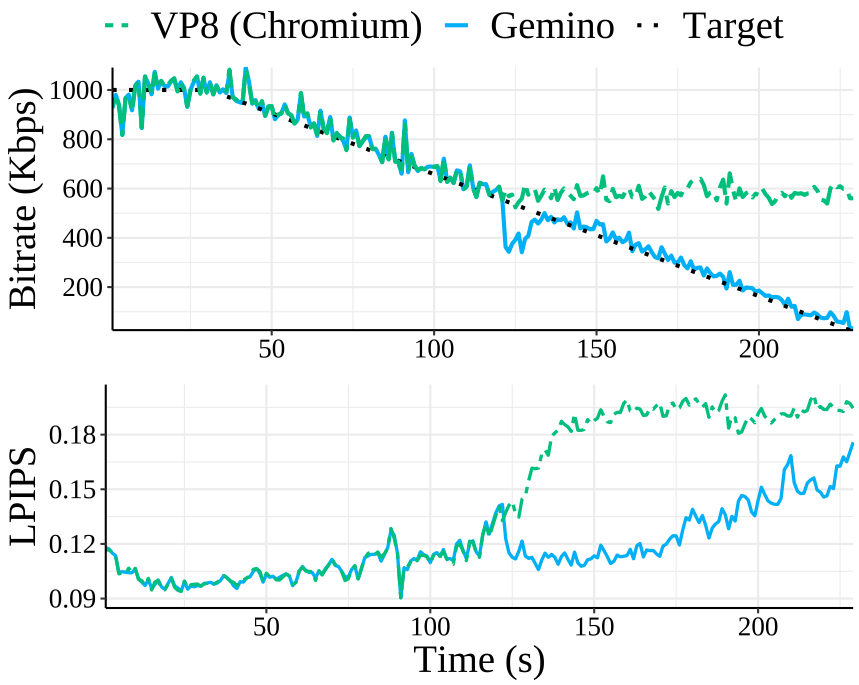


Figure 3-10: Gemino's ability to adapt to a time-varying target bitrate. As the target bitrate reduces, Gemino gradually lowers its PF stream resolution trading off more upsampling and less quality (increased LPIPS) for a reduction in achieved bitrate. VP8, in contrast, lowers the bitrate initially, but once at its minimum quality, it stops responding to the target bitrate.

3.4.5 Adaptation to Network Conditions

To understand the adaptability of Gemino, we explore how it responds to changes in the target bitrate over the course of a video. We remove any conflating effects from bandwidth prediction by directly supplying the target bitrate as a decreasing function of time to both Gemino and the VP8 codec. Gemino uses only VP8 through all bitrates for a fair comparison. Fig. 3-10 shows (in black) the target bitrate, along with the achieved bitrates of both schemes, and the associated perceptual quality [163] for a single video over the course of 220s of video-

time[4]. We observe that initially (first 120s), at high target bitrates, Gemino and VP8 perform very similarly because they are both transmitting just VP8 compressed frames at full (1024×1024) resolution. Once VP8 has hit its minimum achievable bitrate of ∼550 Kbps (after 120s), there is nothing more it can do, and it stops responding to the input target bitrate. However, Gemino continues to lower its PF stream resolution and/or bitrate in small steps all the way to the lowest target bitrate of 20 Kbps. Since Gemino is only using VP8 here, it switches to 512×512 at 550 Kbps, 256×256 at 180 Kbps, and 128×128 at 30 Kbps. This design choice might cause abrupt shifts in quality around the transition points between resolutions. However, Gemino prioritizes responsiveness to the target bitrate over the hysteresis that classical encoders experience which, in turn, leads to packet losses due to overshooting and glitches. As the resolution of the PF stream decreases, as expected, the perceptual quality of Gemino worsens but is still better than VP8's visual quality. This shows that Gemino can adapt well to bandwidth variations, though we leave the design of a transport and adaptation layer that provides fast and accurate feedback to Gemino for future work.

3.5 Summary

This chapter proposes Gemino, a neural video compression scheme for video conferencing using a new high-frequency-conditional super-resolution model. Our model combines the benefits of low-frequency reconstruction from a low-resolution target, and high-frequency reconstruction from a high-resolution reference. Our novel multi-scale architecture and personalized training synthesize good quality videos at high resolution across many scenarios. The adaptability of the compression scheme to different points on a rate-distortion curve opens up new avenues to co-design the application and transport layers for better quality video calls. However, while neural compression shows promise in enabling very low bitrate video calls, it also raises important ethical considerations about the bias that training data can introduce on the usefulness of such a technique to different segments of the human population. We believe that our personalized approach alleviates some of these concerns, but does not eliminate them entirely.

[4]The time series are aligned to ensure that VP8 and Gemino receive the same video frames to remove confounding effects of differing latencies.

Chapter 4

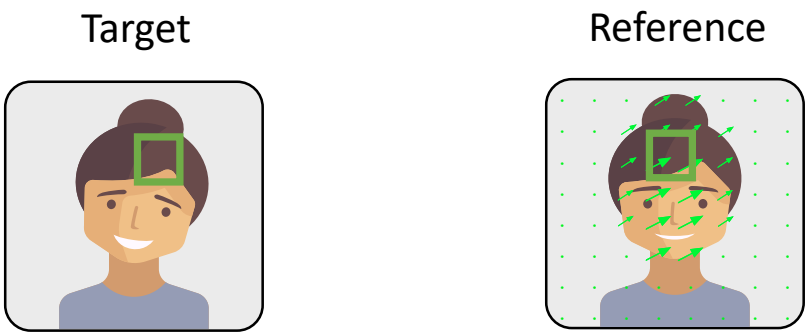
An Attention-Based Design for Gemino

4.1 Motivation

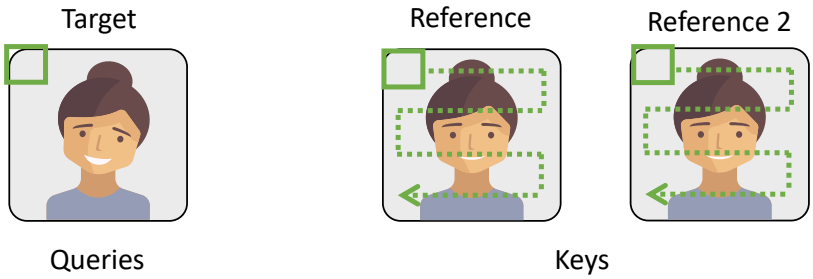
Chapter 3 illustrates that many of the techniques developed for novel-view synthesis using keypoints have catastrophic failures if the reference frame and the target frame differ significantly in their poses. While super-resolution techniques with a series of downsampling and upsampling layers show better low-frequency fidelity and place objects in the right place, they fail to capture high-frequency details such as hair strands and facial texture. Gemino [134] combines both reference-based synthesis techniques and super-resolution in its high-frequency conditional super-resolution design. Specifically, Gemino upsamples a LR target but uses information from a high-resolution reference frame to condition the upsampling. This way, Gemino transfers high-frequency information from the reference frame to the target frame in regions that are similar, but falls back on super-resolution as a lower bound for reconstruction error for regions that are new or very different in the target frame.

While Gemino's design achieves audio-like bitrates with good reconstruction fidelity, it uses only *one* reference frame. This is because the model estimates motion between the reference and the target using optical flow methods and uses the resulting warping field to translate the reference features into the coordinate system of the target prior to decoding. The output from running optical flow maps each region or block of the target frame to its closest match in the reference frame (Fig. 4-1(a)). Though this is an efficient technique for motion estimation, it does not inherently let the model leverage information from multiple reference frames.

Consider the case of a target that has both eyes open and the mouth wide, but needs to choose one of two options for references: one with open eyes and another with an open mouth. An optical flow approach would only be able to leverage one of the two frames, missing out on relevant high-frequency information in either the mouth or eye region. One option would be to add a linear layer after computing individual warping fields based on



(a) Optical flow maps a single target block to a reference block.



(b) Attention computes each target block (query) as a weighted average of all reference blocks (keys).

Figure 4-1: Comparison between optical flow and attention.

each of the two references. Such a linear layer would be responsible for computing weights for which image regions need to focus on which of the two references. Such a weighing layer is outside of the default optical flow formulation. However, this is precisely what the expressiveness of an attention layer lets us do.

Attention [142] is a mechanism to capture correspondence between regions of the target frame and all regions of one or more reference frames. In contrast to optical flow which maps a region of the target to a single region in a reference frame, attention allows you to compute a particular region of the target as a weighted combination of all reference regions (Fig. 4-1(b)). This automatically scales to multiple references since it simply returns weights across all regions of all references. In the above example of the target with open eyes and a wide mouth, attention allows target regions around the eyes to weigh the reference with open eyes more while target regions around the mouth rely more on the other reference with the wide mouth. The precise correspondence is calculated using a region-by-region dot product between the target and the references.

A notable benefit of the dot product operation is that it involves no learning. As a result, an attention-based model trained on a single reference frame can be easily extended to any number of references at inference time. The model simply calculates correspondence

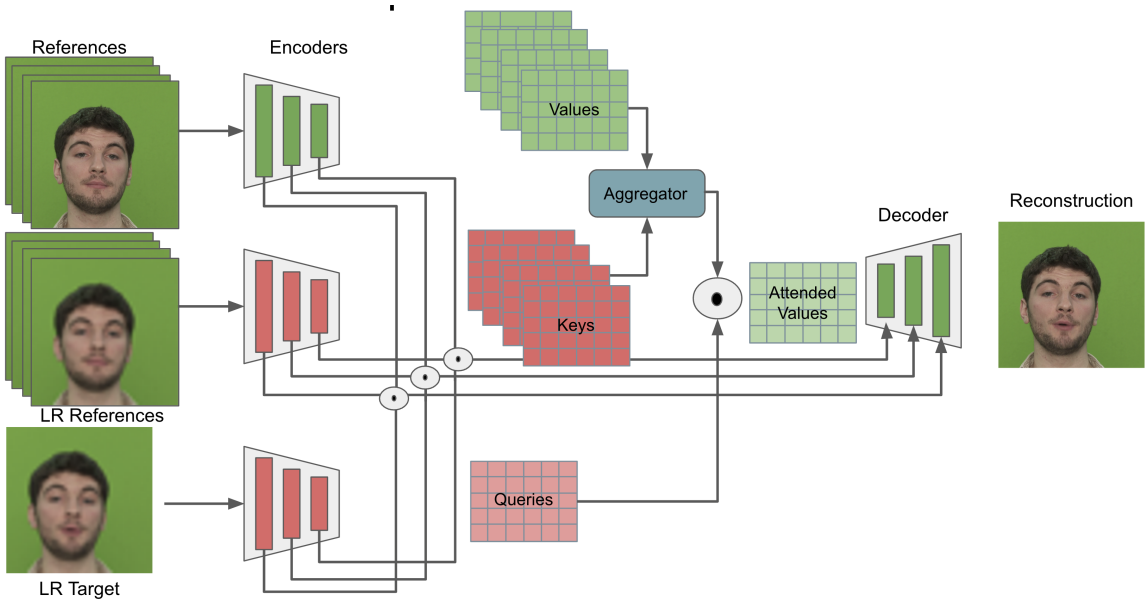


Figure 4-2: Gemino (Attention)'s architecture. The LR target is encoded into query features, while a set of LR and HR reference frame pairs are encoded into their respective key and value feature pairs. The key and value features can be aggregated to smaller dimensions using an aggregator (*e.g.,* concatenation, clustering, PCA). Scaled dot-product attention (denoted by the '.') is computed between query features and each of the key-value pairs to obtain attended HR value features that are then decoded to produce the final reconstruction.

or attention with as many references as provided before running the attended output through the rest of the decoding pipeline. Due to its versatility, we employ attention as an alternative to optical flow for motion estimation in Gemino (Attention), an alternate design to Gemino for high-frequency conditional super-resolution.

4.2 Method

The input to Gemino (Attention) is a set of references in both their LR and HR versions, and a LR target whose HR version we want to reconstruct. Note that the LR and HR images have the same dimensionality because the LR image is re-sized to the same resolution as the HR image using bicubic interpolation. This results in a blurry LR image whose dimensions match that of the crisper HR image. Fig. 4-2 depicts the architecture of our system.

A shared *key-query encoder* encodes the LR references and the LR target into key and query features respectively at multiple resolutions. A separate *value encoder* encodes the HR references into a set of value features also at multiple resolutions. An *aggregator* module (*e.g.,* concatenation) aggregates key-value feature pairs from multiple references into a smaller compact set of key-value pairs. An attention layer computes attention between key and query feature pairs at the coarsest resolution and obtains a set of (coarse)

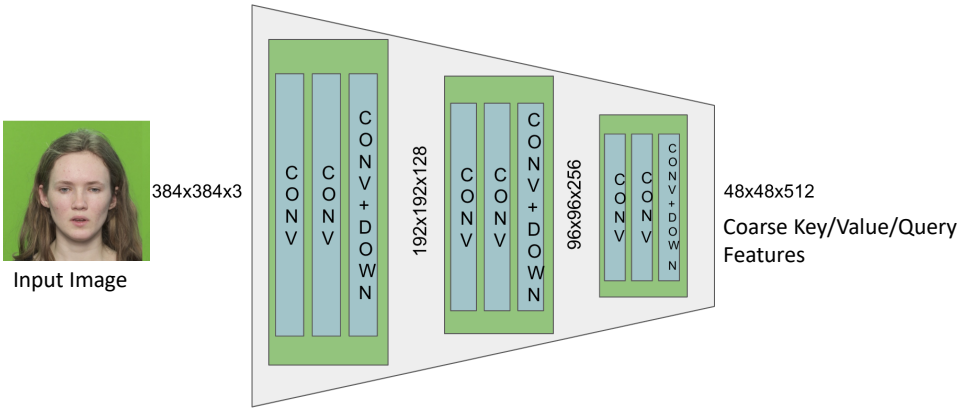


Figure 4-3: Gemino (Attention)'s key-query encoder architecture consisting of two convolutional blocks followed by a downsampling block (convolutions with stride 2) at each resolution. Each block uses ReLU activation. The encoding layers convert the 384×384×3 RGB image into coarse key, query or value features of size 48×48×512. The decoder architecture mirrors the encoder, but uses 2D upsampling layers instead of the downsampling layers.

attended value features by mapping the keys to their corresponding values. This process is repeated at higher resolutions (or finer levels) to provide skip connections that are leveraged during decoding to preserve high-frequency content. The attended value features at the coarsest level are then decoded, along with attended skip connections at higher resolutions, to produce the final reconstructed image.

We describe in detail the key-query encoder, the value encoder and decoder in §4.2.1. A strawman version of attention that uses a single reference image is described in §4.2.2. We extend this strawman to multiple references in §4.2.3, and finally describe how we leverage skip connections and attention at multiple resolutions to preserve high-frequency content in §4.2.4.

4.2.1 Feature Encoding and Decoding

To capture information from the LR and HR images at multiple resolutions in a richer space than their per-pixel RGB representations, we first encode them into the feature space. We then perform attention before decoding the same features.

Key-Query Encoder. The LR references and the LR version of the target frame are encoded using a shared key-query encoder. Since our attention mechanism computes correspondences between the keys and the queries, it is best that their features are semantically aligned, and so, we use a shared encoder to improve that likelihood. The key-query encoder consists of three convolutional layers each with stride 1. Each of these layers is paired with a downsampling layer that runs convolutions of stride 2. Thus, each downsampling

layer reduces each spatial dimension by a factor 2, decreasing the total spatial blocks by 4x. The convolutions use a 3×3 kernel and 'same' padding to capture information across regions of the input images. Each convolutional layer also uses a ReLU activation. Fig. 4-3 diagrammatically describes this encoder design. Since we use 384×384 images in our evaluation, the encoder produces features with spatial dimensions of 192×192, 96×96 and 48×48 at each intermediate layer with 128, 256 and 512 features respectively. The coarsest features of dimensions 48×48×512 form the key and query features that are used in the attention layer described in §4.2.2. We extend this structure to multiple resolutions using the features at the intermediary layers in §4.2.4.

Value Encoder. The architecture of the value encoder is the same as the key-query encoder. In other words, the value encoder also uses three convolutional layers with stride 1, each of which is paired with a downsampling layer that reduces each spatial dimensions by 2×. The final output after all three convolutional layers are the coarsest value features of dimensions 48×48×512 which are used in the attention layer prior to decoding. However, unlike the key-query encoder, the value encoder operates on high-resolution references and thus, encodes high-frequency content associated with the reference images.

Decoder. The decoder's design nearly mirrors that of the value encoder since it is designed to produce an RGB image from the attended (value) features. Specifically, the decoder consists of three convolutional layers each with stride 1. Each of these layers is paired with a two-dimensional upsampling layer that performs a bilinear interpolation to increase the spatial dimensions of the feature by 2× along each axis. The convolutions use a 3×3 kernel and 'same' padding to combine information across regions of the input images. Each convolutional layer also uses a ReLU activation. The architecture is a mirror of the encoder shown in Fig. 4-3 with Upsampling layers replacing the down blocks. The decoder further uses skip-connections produced by attending appropriately to the encoded value features at the resolutions in intermediary layers of the encoder. We motivate the use of these skip connections further in §4.2.4.

4.2.2 Attending to a Single Reference

As described in §4.2.1, the key-query encoder produces query features from the LR target frame and key features from the LR reference frames, while the value encoder encodes the HR reference frame into value features. The next step is to compute correspondence between these query and key features. For simplicity, we assume that we have exactly one reference frame which ensures that the query, key, and value features are all of identical dimensions. We further assume that attention happens only at one resolution at the coarsest level with keys, values and queries each of dimensions 48×48×512. We extend this design to

multiple references in §4.2.3 and multi-resolution attention with skip connections in §4.2.4.

To compute correspondence between the query and key features, we use the scaled dot-product attention layer [142, 12]. Specifically, we compute:

$$\text{Attended Values(Q, K, V)} = \text{softmax}\Big(\frac{QK^T}{c}\Big)V$$

where Q, K, and V denote query, key and value features respectively, and c is a learnt scaling parameter. This layer computes the inner-product QK^T between each query element against each key element, and obtains a softmax over it to produce a weighted average of value elements across all locations that contribute to a particular query location. Effectively, this layer captures the similarity between the query and the key or learns how to combine different regions of the reference to produce a particular region of the target. Since we ultimately want to use the HR features in the decoding procedure, this similarity, though computed on LR reference features, is mapped to HR value features from the reference via the multiplication with value features V.

Consider a concrete example for query features with $d=512$ filters. If the 48×48 spatial dimensions are flattened, the query Q is $2304\times d$ in size. Since the key-query encoder is shared and the value encoder's architecture is identical to that of the key-query encoder, the encoded values V and keys K from a single reference are also $2304\times d$ in size. This produces a similarity matrix $S = QK^T$ of dimensions 2304×2304. Each entry in the similarity matrix S captures the correlation between a particular query location and a particular key location by summing the inner product on a filter-by-filter basis. This is repeated for every combination of query and key location to obtain the entire 2304×2304 matrix. The softmax function is applied to S scaled by a learnt parameter c to ensure that the similarities sum up to 1 before multiplying with the value matrix V. Mathematically, if k_i,v_i,q_i for $i\in\{1,...,2304\}$ denote the i^{th} key, value or query feature vector with their d filters, this process produces attended values A of dimensions $2304\times d$. The i^{th} attended feature vector (with d filters) is given by

$$A_i = \sum_{x=1}^{2304} q_i.(k_x)^T \cdot v_x$$

where we have ignored the softmax and scaling parameter c for simplicity. The attended values A are reshaped into $48\times48\times d$ to preserve spatial locality information prior to decoding them into final reconstruction.

Note that we choose to use the simplest version of attention by leveraging a single

dot-product attention layer with c being the only learnt parameter. Multi-headed attention [142] is more commonly used in transformer architectures, and is known to be much more powerful. However, we did not see significant gains in our evaluations, and thus chose to avoid the increased computational overheads.

4.2.3 Attending to Multiple References

Using attention even on a single reference image allows for improved capabilities in combining information from multiple image regions in contrast to optical flow that moves a single source location to a single target location. However, the versatility of attention is put to its true use with multiple reference images. To extend the design described in §4.2.2 to multiple images, we alter the attention layer to now compute correspondences across many more features from multiple references.

Specifically, we assume that we have n different reference images, and we have LR and HR versions of each reference. Each of these n references are encoded using the query-key encoder and the value encoder to produce n different key and value features, each $2304 \times d$ in size. We simply concatenate the key features across all reference frames to produce an expanded matrix of key features $K^{(n)}$ of size $2304n \times d$. Similarly, we concatenate the value features across all reference frames to produce an expanded matrix of value features $V^{(n)}$ also of size $2304n \times d$. The query features are still encoded from a single target LR image; Q is still $2304 \times d$.

The new similarity matrix $S^{(n)} = Q[K^{(n)}]^T$ is now of dimensions $2304 \times 2304n$. Yet, each entry in the new similarity matrix still captures the correlation between a particular query location and a particular key location by summing the inner product over the same d filters as with the single reference image case. The only difference is that there are more key locations ($n\times$ more) to compute this correlation over, and consequently a larger similarity matrix. When multiplied with the larger value matrix $V^{(n)}$, this produces final attended values of the same $2304 \times d$ as before which are decoded to produce the reconstructed image. Mathematically, we simply sum over more terms to produce attended values A of dimensions $2304 \times d$. Specifically, the i^{th} attended feature vector is now given by

$$A_i = \sum_{x=1}^{2304n} q_i . (k_x)^T \cdot v_x$$

This reveals a very powerful detail about the attention-based design in our model. The attention layer, even if trained on a single reference image to learn its scaling parameter c, can be extended at inference time to use multiple reference images *without retraining*. This is because the sizes of the key and value pairs are abstracted away in the dot-product; only

the query feature dimensions and the final attended value features need to be maintained for the model pipeline from Fig. 4-2 to work between train and test time. Note that we do not re-scale the learnt parameter c based on the number of references used at inference time. This parameter, as per the original attention layer [142], depends on the number of filters or the dimension d which remains unchanged between the single reference and multi-reference versions of our design.

4.2.4 Preserving High-frequency Content

Multi-resolution Attention. The attention mechanism described above uses an example resolution of 48×48 for its query, key and value features. However, in practice, the spatial resolution and channel dimensions (d) for the features can have an outsized impact on the granularity of these features and subsequently, the quality of the final reconstruction. As described so far, one solution would be to run attention once on the coarsest (last) level of features obtained from the query, key, and value features. In our evaluations with 384×384 images and encoders with three downsampling layers, this would mean spatial dimensions of 48×48. While this makes the computationally intensive attention operation tractable, such a resolution is too coarse to realistically preserve any high-frequency content associated with the images.

Instead, we leverage its skip connection design popularized by U-Nets [124] to improve the fidelity to high-frequency content in Gemino (Attention). Since we already have an encoder and decoder structure, it is relatively straightforward to extend our design to use skip connections. Specifically, we obtain encoded query, key and value features from each intermediary level of the key-query encoder and value encoder. This results in features of spatial dimensions 192×192 and 96×96 in addition to the coarsest 48×48. We compute attention or correspondence between each of these levels' query and key features to produce intermediary attended value features. While the coarsest level's attended features are used as the input to the decoder, the remaining levels' attended features are progressively fed in as skip connections to improve the fidelity to high-frequency content in the final reconstruction. We deem this architectural element "multi-resolution attention" and evaluate its benefits in §4.3.2.

Block-based Attention for Reducing Compute. Though the skip connections help with improving fidelity, running attention at resolutions as high as 192×192 is simply impractical. This is because the dot product involved is effectively computed between every spatial query and key location resulting in a similarity matrix of size 192×192×192×192. To overcome this issue, we break our feature space into smaller patches and compute attention only within that patch. This idea is inspired by ViT [51], but modified to account for the fact that we are only synthesizing talking heads.

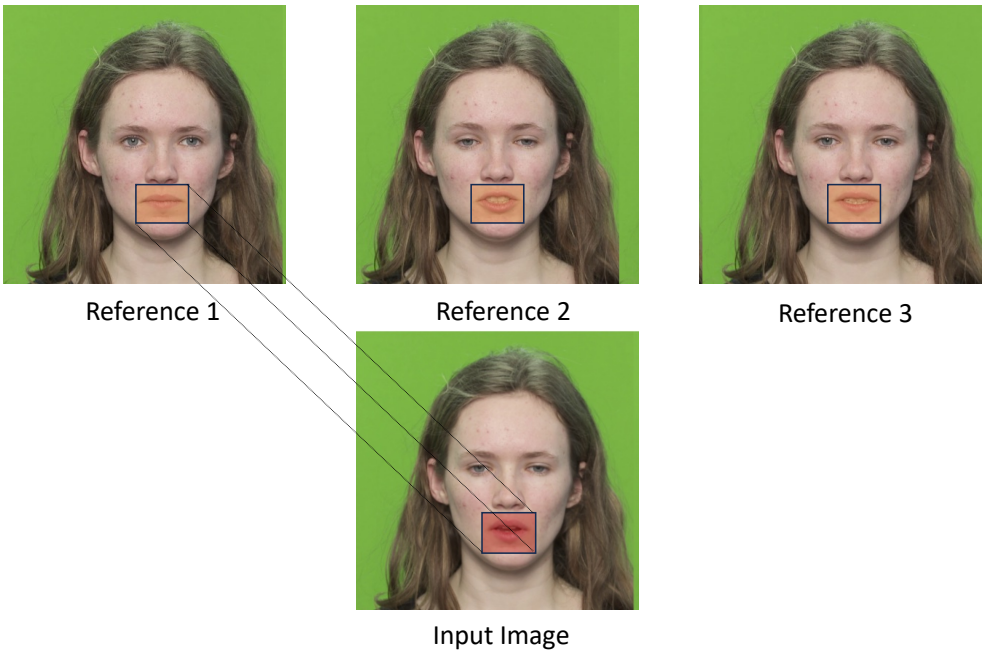


Figure 4-4: Gemino (Attention)'s block-based attention design to reduce the computational overheads of computing pairwise correspondence between all features at higher resolutions. Instead, the features are broken into 12×12 blocks and attention is only computed between the 144 query, key, and value features within the same block (identified by location index) across all references. For example, this ensures that query features corresponding to the mouth use a narrow set of features around the mouth region from all three reference frames when computing attention.

In the design of Gemino (Attention), we break features that are spatially large into 12×12 blocks and only compute attention within the 144 locations inside each of these blocks and across the 144 locations in the same block of all the reference image features. In other words, query features of dimensions $192\times192\times d$ (where d is the filter or channel dimension) are broken into 256 separate $12\times12\times d$ blocks with their indices identifying which region in the original 192×192 they came from. We repeat this across all n reference frames' key and value features to generate n separate key and value features for each 12×12 block index. Then, we pick a particular block index, and compute attention between the $144\times d$ query features and its $144n\times d$ key and value features for that block. As seen in Fig. 4-4, one can visualize this as first narrowing down to a small set of features mapping to the mouth region, and then leveraging information on the mouth's orientation across n frames to reconstruct the mouth in the desired orientation for the target. This avoids wasting compute on attention between areas close to the mouth and areas irrelevant to the mouth such as the eye or hair that are likely to result in weak or no correlations. This block-based attention is similarly repeated to attend to features in regions focused on the ear, eye, nose, *etc.*.

4.3 Evaluation

4.3.1 Setup

Dataset. To evaluate Gemino (Attention), we use the TCDTimit [63] dataset that consists of 1080p videos that are recordings of trained actors who read a script facing a camera against a green screen. The videos are short clips of duration less than 10 seconds and are at a framerate of $\sim$30 FPS. The videos are of high-quality, encoded at bitrates above 50 Mbps. Out of the 60 actors, we choose 51 to form our training set and nine people to form our test set. Each person has 96 videos. Prior to training, we crop a square frame of size 384×384 in the center. The low-resolution (LR) input is either of size 48×48 or 96×96 depending on whether it is a 8× or 4× upsampling task. In both cases, the LR frames have been resized to 384×384 by bicubic upsampling prior to use in the training or inference pipeline. We run inference on every frame of every test video of every person, but for training, we sample 1000 pairs of reference and target frames per video. Note that the people in the train and test dataset are completely different; unlike Gemino, this approach does not leverage per-person finetuning for improved reconstruction fidelity.

Baselines. We compare the following approaches to understand the benefits that Gemino (Attention) provides compared to existing solutions.

- **Single-image Super-resolution (SISR)**: A simple encoder and decoder pair that encodes the LR target image and decodes it into the HR target image. This approach does not use any HR reference frames.
- **Gemino**: A version of the Gemino [134] model that does 8× upsampling. Gemino operates directly on the LR input (48×48) without resizing it using bicubic upsampling to 384×384. We use this only to provide a baseline for compute overheads (and not visual fidelity) since Gemino was trained on a very different dataset with personalization benefits that we do not explore in this attention-based model.
- **Coarse Attention with Single Reference Image (Coarse Attn.)**: This approach, like Gemino, encodes the LR and HR frames into a set of encoded features. However, instead of computing optical flow to warp the HR features, it computes attention on the coarsest (or smallest spatial dimension) features of the HR frame with those of the LR frame. This attention is run over the entire spatial dimension (48×48) of these encoded features. The decoding pipeline does not use any of the skip connections described in §4.2.4.
- **Gemino (Attention)**: Our contribution that builds upon the previous "Coarse Attention" model by running attention at multiple resolutions using the intermediary outputs of the encoder's layers when applied to the HR and LR reference and

target frames. Each of these intermediary attended outputs is provided to the decoder as skip connections to improve its high-frequency fidelity. To ensure that this approach scales as the number of reference frames is increased, attention is computed over smaller blocks to focus the attention mechanism over small regions of all the references. This model is trained using a single reference image randomly sampled from the same video as the target image, but is used at inference time with upto 10 reference images in our evaluations. In such cases, the encoded features from all the references are provided as key-value pairs to the attention module that decided how to appropriately weigh them when computing features to decode into the target image.

Model and Training Procedure. We train all models on the entire corpus of train videos for 1M steps. Gemino (Attention) is trained with a batch size of 8 with random rotational augmentations of ninety degrees to improve its robustness. The learning rate starts at 0.0001 and decays by a factor of 0.46 halfway. Our model is trained with both an L1-loss on the pixel space between the ground truth and the reconstruction, as well as a feature matching loss from VGG [76] that improves the sharpness of the output produced. Both these losses are weighed equally.

Visual Quality Metrics. We compute the average Peak Signal-to-Noise Ratio (PSNR), the Structural Similarity Index (SSIM) [151], and the Learned Perceptual Image Patch Quality Metrics (LPIPS) on the reconstructed frames from each of the above baselines and the ground-truth. PSNR is reported as the mean-square error of pixels in each frame averaged across all test frames of a given video, measured on the RGB scale. SSIM is reported as the average (across frames) of the structural similarity between the reconstruction and its high-resolution counter-part. For SSIM and PSNR, a higher value denotes better reconstruction quality. LPIPS is reported as a distance metric in the VGG [133] feature spaces of the reconstructed image and its high-resolution ground-truth. Lower LPIPS scores correspond to higher fidelity. We also show visuals to provide qualitative comparisons across approaches.

Computational Overhead Metrics. In addition to assessing the fidelity of the reconstructions to their ground truth, we also measure the computational overhead of each of the models. This is important because attention is an expensive operation since it computes large matrix dot products. We measure the the size of the model in terms of its number of parameters, its computational complexity as reflected by the number of floating point operations (FLOPS) it needs, as well as the time it takes to run inference on a single frame on a V100 GPU. We ignore any one-time operations such as encoding time of the references for Gemino (Attention) since this will likely be done once, and then reused for subsequent attention computations.

4.3.2 Results

	PSNR (dB) ↑			SSIM (dB) ↑			LPIPS ↓		
Scheme	Avg.	P5	P1	Avg.	P5	P1	Avg.	P95	P99
SISR	30.98	28.72	28.56	8.14	6.77	6.69	0.17	0.21	0.22
Coarse Attention	30.98	28.73	28.55	8.07	6.70	6.62	0.17	0.20	0.21
Gemino (Attention): 1 ref	32.69	29.86	29.39	9.55	7.19	7.04	0.11	0.15	0.16
Gemino (Attention): 2 refs	32.67	29.83	29.42	9.50	7.25	7.08	0.11	0.15	0.16
Gemino (Attention): 5 refs	**32.84**	**29.86**	**29.46**	**9.62**	**7.33**	**7.13**	**0.10**	**0.14**	**0.15**

Table 4-1: Performance improvements from using multiple references with Gemino (Attention) running over 24×24 blocks over Single-Image SR (SISR) and attention that runs at the coarsest level alone (Coarse Attention) at an 8× upsampling task. SISR and Coarse Attention miss high-frequency information while Gemino (Attention) retains it.

i

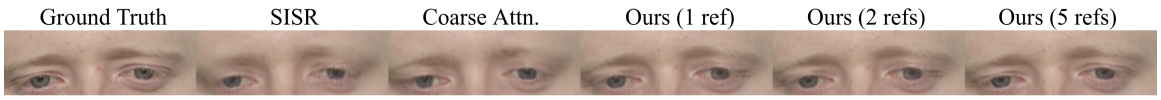
Main Takeaway. We compare all of the baselines against Gemino (Attention) when upsampling 8× from a 48×48 frame to a 384×384 frame and present a quantitative summary in Tab. 4-1. We show visual samples in Figures 4-5 and 4-6 to provide qualitative examples. Overall, Gemino (Attention) outperforms other baselines, and also provides benefits in reconstruction quality as the number of references increases. Specifically, it maintains high-frequency fidelity when compared to SISR and Coarse Attention, but also performs better in particular regions of the face and torso with improved information from multiple references. We break these results down in subsequent paragraphs.

Comparison with Single-Image SR. We observe in Tab. 4-1 that single-image SR (SISR) achieves on average 1.7 dB less PSNR, 1.41 dB less SSIM, and 0.06 more LPIPS than Gemino (Attention) with one reference frame. A similar difference of 0.06 manifests with LPIPS on the worst 5% and 1% of test frames. Without a reference frame to provide high frequency information, SISR's upsampling layers are unable to recover the details associated with individuals' facial features. As seen in Figures 4-5 and 4-6, the SISR output misses the freckles and acne associated with both individuals' foreheads. Its reconstruction of both individuals' hair also lacks detail. This is unsurprising since the low-resolution input frame lacks high-frequency detail. Further, these models have not been trained with frames of the test individuals and thus, cannot encode such information into their weights.

Benefits of Multi-resolution Attention. Tab. 4-1 shows that "Coarse Attention" does not perform better than SISR. This suggests that running attention at the coarsest level with encoded reference features fails to capture the same high-frequency details that SISR misses. This is reflected in the visual samples in Figures 4-5 and 4-6 too; Coarse Attention

(a) Pool of Five References

(b) Reconstruction of Different Schemes.



(c) Zoomed-In View of Reconstruction of Different Schemes.

Figure 4-5: Reconstruction quality of different approaches on a specific frame using upto five references. Unlike Gemino (Attention), SISR and Coarse Attention miss high-frequency information associated with freckles and acne on the face. As the number of references for Gemino (Attention) is increased, the reconstruction of the eyelashes and the folds of the eyes improves as seen in the zoomed-in version.

misses the same freckles and acne that SISR misses on the foreheads. This is because the encoded features at the coarsest level of the encoder are too low-resolution (48×48) to retain enough high-frequency information from the reference frame that can later be leveraged through attention before the decoding pipeline.

However, when that same information is passed through a series of skip connections in Gemino (Attention), the reconstruction quality improves significantly even with one reference frame. Specifically, the skip connections provide reference features at multiple resolutions via attention computed on each intermediary encoder layer's outputs. The decoder then uses these reference features at the appropriate resolution to recover the high-frequency details associated with the individual. This improved architecture preserves the acne and freckles in the visual samples produced by Gemino (Attention) in Figures 4-5 and 4-6.

SR Task Difficulty. In Tab. 4-2, we compare existing baselines to Gemino (Attention) at a 4× upsampling task from 96×96 to 384×384. Like the 8× upsampling case, we observe that Gemino (Attention) outperforms SISR and attention at the coarsest level alone due to the lack of sufficient high-frequency information in both approaches. As the number

(a) Pool of Five References

(b) Reconstruction of Different Schemes.

(c) Zoomed-In View of Reconstruction of Different Schemes.

Figure 4-6: Reconstruction quality of different approaches on a specific frame using upto five references. Unlike Gemino (Attention), SISR and Coarse Attention miss high-frequency information associated with freckles and acne on the face. As the number of references for Gemino (Attention) is increased, the reconstruction of the eyelashes and the eye gaze improves as seen in the zoomed-in version.

	PSNR (dB) ↑			SSIM (dB) ↑			LPIPS ↓		
Scheme	Avg.	P5	P1	Avg.	P5	P1	Avg.	P95	P99
SISR	33.61	31.23	31.02	9.76	8.10	8.02	0.12	0.14	0.15
Coarse Attention	33.61	31.18	30.97	9.78	8.16	8.05	0.12	0.14	0.15
Gemino (Attention): 1 ref	34.57	32.01	31.31	10.77	8.54	8.40	0.09	0.11	0.13
Gemino (Attention): 2 refs	34.55	32.04	31.31	10.73	8.60	8.44	0.09	0.11	0.13
Gemino (Attention): 5 refs	34.63	**32.05**	31.32	10.78	8.64	8.44	0.09	0.11	0.13
Gemino (Attention): 10 refs	**34.71**	31.98	**31.32**	**10.86**	**8.65**	**8.46**	**0.09**	**0.11**	**0.13**

Table 4-2: Performance improvements from using multiple references (15 frames apart) with Gemino (Attention) running over 12×12 blocks over Single-Image SR (SISR) and attention that runs at the coarsest level alone (Coarse Attention) at an 4× upsampling task. The trends across approaches are similar to 8× upsampling but the reconstruction quality is much better since the starting resolution is higher.

Scheme	Params	FLOPs	V100 Inference
SISR	11.66M	191B	11.65ms
Coarse Attention	18.60M	385B	285.81ms
Gemino	82.41M	82B	14.87 ms
Gemino (Attention) @ 12×12			
1 reference	20.22M	481B	29.72ms
2 references	20.22M	655B	30.15ms
5 references	20.22M	1178B	47.38ms
10 references	20.22M	2049B	66.14ms
Gemino (Attention) @ 24×24			
1 reference	20.22M	540B	31.67ms
2 references	20.22M	774B	47.23ms
5 references	20.22M	1476B	98.70ms

Table 4-3: Computational overheads of Gemino (Attention) at block sizes of 12×12 and 24×24 with different number of reference frames in comparison to single-image SR, attention at the coarsest level and Gemino's optical flow version. Parameters, floating point operations (FLOPS), and inference time on a V100 GPU are measured in millions, billions, and milliseconds respectively.

of references is increased to ten, we observe a marginal improvement in visual quality as reported by the PSNR, SSIM and LPIPS values for the average and worst 5% and 1% of frames. Since 4× upsampling is generally easier than 8×, all of the reported numbers are much better than those reported in Tab. 4-1.

Computational Overheads. Though using attention with multiple references instead of optical flow improves the visual quality of generated frames, computing attention involves multiplication of large matrices that imposes severe overheads. As seen in Tab. 4-3, single-image SR takes only 11.65 ms to run inference on a 384×384 frame, and has far fewer parameters and FLOPs in comparison to its attention counterparts. Gemino [134] has a lot more parameters ($\sim$ 82M) because it has a separate keypoint extraction and motion estimation pipeline. However, because many of these operations run at lower resolutions in its multi-scale architecture, this does not cause an explosive increase in FLOPs or inference time.

All of the attention versions of Gemino and coarse attention have fewer parameters than Gemino's optical flow version because we use dot-product scaled attention for motion estimation which only has one parameter regardless of the attention block size and the input dimensions. However, as the block size increases for a fixed number of references, the FLOPs for Gemino (Attention) increases because each attention operation computes matrix multiplication over larger spatial regions. This consequently manifests as an increase in V100

inference time. As the number of references increases, attention becomes more expensive (increased FLOPs and inference time) once again because of more matrix multiplications. Note that this increase is not quite linear because we only measure the inference time for the attention and decoding pipeline, and assume that the encoding process from multiple references into features is only done once at the beginning of the video call.

Since attention at the coarsest level computes attention over 48× 48 blocks, its inference time is quite high due to prohibitively large matrix multiplication operations despite the fact that the encoded features themselves are only 48×48 (only one attention block as a result). Note that the FLOPs for the coarse attention approach is lower than Gemino (Attention) running attention over multiple separate 12×12 or 24×24 blocks. Yet, the size of the attention matrix with coarse attention at 48×48 itself is 16 times larger than Gemino (Attention) with 24×24 blocks. This could result in more memory bottlenecks [48] when creating large matrices causing the overall inference time with coarse attention to be nearly 3× larger than Gemino (Attention).

4.3.3 Ablation

Number of Reference Frames in Gemino (Attention). Tab. 4-1 shows small improvements in reconstruction quality for the average and worst 5% and 1% of frames as the number of references in Gemino (Attention) is increased. Specifically, we observe 0.15 dB increase in PSNR, 0.07 dB increase in SSIM and 0.01 decrease in LPIPS when using five references. These quantitative differences are small because the TCD-Timit dataset [63] is highly curated to involve actors speaking a script in solely front-facing poses with minimal movement. We anticipate the differences being more substantial in datasets that involve more movement and a range of poses.

Despite these small quantitative differences, we observe visual differences with five references in Figures 4-5(c) and 4-6(c). Specifically, Fig. 4-5(c) shows that the reconstruction of the eyelashes and the folds of the eyes is a lot more accurate with five references where the latter three have open eyes when compared to using one or two references with closed eyes. Similarly, Fig. 4-6(c) shows a more accurate reconstruction of the eye gaze and lashes with five reference frames that have different gazes than with two references with very similar gazes.

Impact of Attention Block Size. We show the effect that the attention block size has on reconstruction quality with five references when upsampling by 8× in Tab. 4-4. The attention block size determines the size of the region over which attention is computed or how large of a reference frame area is used to compute weights for a particular target location. Tab. 4-4 suggests that as the attention block size increases, the reconstruction quality (LPIPS) improves for the average and worst frames in the corpus. As we move from 6×6

Scheme	PSNR (dB) ↑			SSIM (dB) ↑			LPIPS ↓		
	Avg.	P5	P1	Avg.	P5	P1	Avg.	P95	P99
Gemino (Attention) @ 6×6	32.30	29.33	28.97	9.13	7.37	6.99	0.12	0.16	0.18
Gemino (Attention) @ 12×12	32.21	29.33	28.90	9.04	7.16	6.89	0.12	0.15	0.18
Gemino (Attention) @ 24×24	**32.84**	**29.86**	**29.46**	**9.62**	**7.33**	**7.13**	**0.10**	**0.14**	**0.15**

Table 4-4: Impact of attention block size in Gemino (Attention) when using five references. A larger block size allows the attention module to leverage information across a wider region when computing correspondence, leading to better performance.

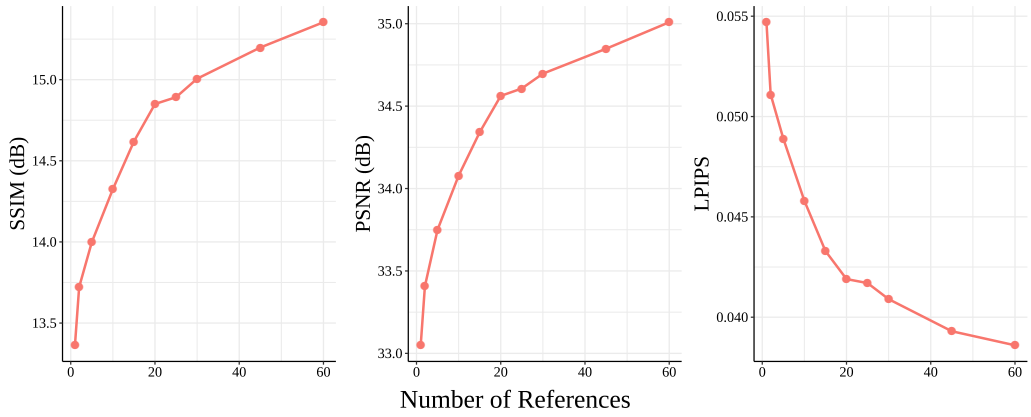


Figure 4-7: Visual quality on a more diverse dataset with increasing number of references. We see diminishing returns as the number of references is increased, but over 1 dB improvement in PSNR and 20% reduction in LPIPS in going from 1 to even 10 references.

blocks to 24×24 blocks, we observe an improvement of 0.54 dB in PSNR, 0.49 dB in SSIM and 0.02 in LPIPS for the average frame. Though we anticipate some improvements from increased reference information at higher block sizes, attention with 48×48 blocks becomes prohibitively expensive to the point of not being able to incorporate multiple reference frames. At such large blocks, the attention weights also tend to average out more causing blurring effects similar to "Coarse Attention". As a result, for 8× upsampling, we use 24×24 blocks.

4.3.4 Results on a More Diverse Dataset

To show the benefits from multiple references on a more diverse dataset, we train our attention model and evaluate it on a separate proprietary dataset consisting of individuals in front-facing positions. This model used the same training parameters as described in §4.3.1, but was trained for 3M steps. Unlike TCDTimit [63], this dataset does not consist of actors with scripts with minimal head movement. In contrast, the individuals in this harder dataset tend to move around more and exhibit variety in their poses. Their backgrounds

also differ across videos and do not consist of a single green screen. However, the videos are not of such high quality and tend to be close to a few Mbps in their bitrates. In an effort to show results at higher numbers of references (~ 60), we train and test the model that uses "Coarse Attention" mechanism without multi-resolution attention or the skip connections described in §4.2.4. This is because running attention at multiple resolutions incurs prohibitively high memory overheads as the number of references is increased.

Fig. 4-7 shows the improvement in visual quality on the above described dataset as the number of references is increased when upsampling $8\times$ to a final resolution of 384×384. Specifically, on this dataset, we observe a nearly 1 dB improvement in both SSIM and PSNR and a 20% decrease (0.01) in LPIPS in going from 1 to even 10 references. This is in contrast to the $4\times$ upsampling case with TCDTimit shown in Tab. 4-2 that sees very slight improvements in going from 1 to 10 references. Further, we see diminishing returns from increasing the references all the way to 60 with this dataset. While there is a big improvement in going from 1 to about 20 references, subsequent increases from 20 to 30 or 45 references show far less improvement in video quality. Note that these reference frames are consecutive and we run attention only at one-resolution, yet there are significant improvements from using more references when compared to TCDTimit [63]. We anticipate even more gains with attention at multiple resolutions and reference frames that are further apart.

4.4 Summary

In this chapter, we develop an alternate design for the high-frequency conditional super-resolution technique proposed by Gemino [134] that leverages attention for motion estimation. Our approach is able to utilize high-frequency information from a diverse set of references to better predict the target frame than with one single reference. The scaled-dot product attention layer also allows flexibility in picking the exact number of references at inference time even though the model is trained for only one reference frame. This flexibility allows for the design of new adaptation algorithms that dynamically choose the number and specific reference frames that trade off accuracy for compute when deployed on the receiver side in settings such as video conferencing. We leave the design of such algorithms as well as optimizations to speed up our attention-based model architecture to future work.

Chapter 5

Reparo: A Loss-Resilient Generative Codec

5.1 Motivation

The info-graphic depicting broadband speeds across the globe in Fig. 1-2 suggests that regions of Europe and North America have such high broadband speeds (on the order of hundreds of Mbps) that video conferencing must be seamless. Yet, users of video conferencing applications in these regions are unsatisfied with their experience. Specifically, 30% of users report that video quality issues are their biggest pain point with video conferencing calls [20]. This happens despite the high broadband speeds in these regions because such speeds capture an *average* number while the frustrating video glitches and audio outages happen due to rare *tail events* such as 1s of poor network conditions every few minutes. Such outages are hard to predict since they can occur anywhere from the first mile (*e.g.,* interference on the WiFi connection) to somewhere in the core of the network (*e.g.,* damages to fibers). Simply over-provisioning the network is extremely cost ineffective for the average case. Further, it may still cause issues under more extreme outages.

Video conferencing experience is also particularly sensitive to bursty losses [97, 96] where a sequence of packets, and consequently, a sequence of frames are lost. In addition to not being able to render the lost frames, applications often encounter longer extended glitches past the loss duration itself. Such bursty losses are not uncommon; they can be induced by congestion from the video encoder exceeding the available bandwidth [77], bursty cross traffic [73] or even link fading on wireless connections [145].

Gemino uses a novel high-frequency condition super-resolution pipeline to support video conferencing at very low audio-like bitrates. However, Gemino is not robust under packet losses because it requires a reliable low-resolution video stream to upsample from. Any packet loss in that stream causes the same frame corruption (albeit at low-resolution) and video glitches. The reason for this is the temporal dependency across video frames. Video frames use delta encoding wherein they encode frames based on differences between

them to exploit redundancy. This allows video to be compressed by orders of magnitude more than simple image compression but also makes it brittle under packet loss. One corrupt frame (due to packet loss) makes it impossible to decode any P-frames or predicted frames encoded based on it until the decoder state is reset using a new I-frame or keyframe that is encoded independently with just image compression. However, such a reset typically takes one round-trip-time (RTT) or a few frames before the receiver can request the sender to send a keyframe. In the meantime, the receiver is forced to contend with a stall and simply render the last successfully decoded frame.

Many existing video applications use retransmissions or forward error correction (FEC) to improve robustness against packet loss. However, retransmissions take on the order of few RTTs and can be very slow in real-time settings such as video conferencing. On the other hand, FEC approaches need careful tuning to achieve the right amount of redundancy without under-utilizing the network under good network conditions. Instead, in this chapter, we propose a new generative codec that completely eliminates temporal dependency across frames. Specifically, our solution leverages a token-based architecture that encodes frames based on a dictionary or codebook that captures the visual world. Our codec compresses by simply transmitting token indices for small patches on the entire frame. Any missing tokens from packet loss can be reconstructed based on correlations across the received tokens and their locations. As a result, our approach is able to render frames regardless of what the loss level is and prevent any stalls due to non-rendered frames.

5.2 Reparo Design

5.2.1 Overview

Reparo is a generative loss-resilient video codec specifically designed for video conferencing. As shown in Fig. 5-1, 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.

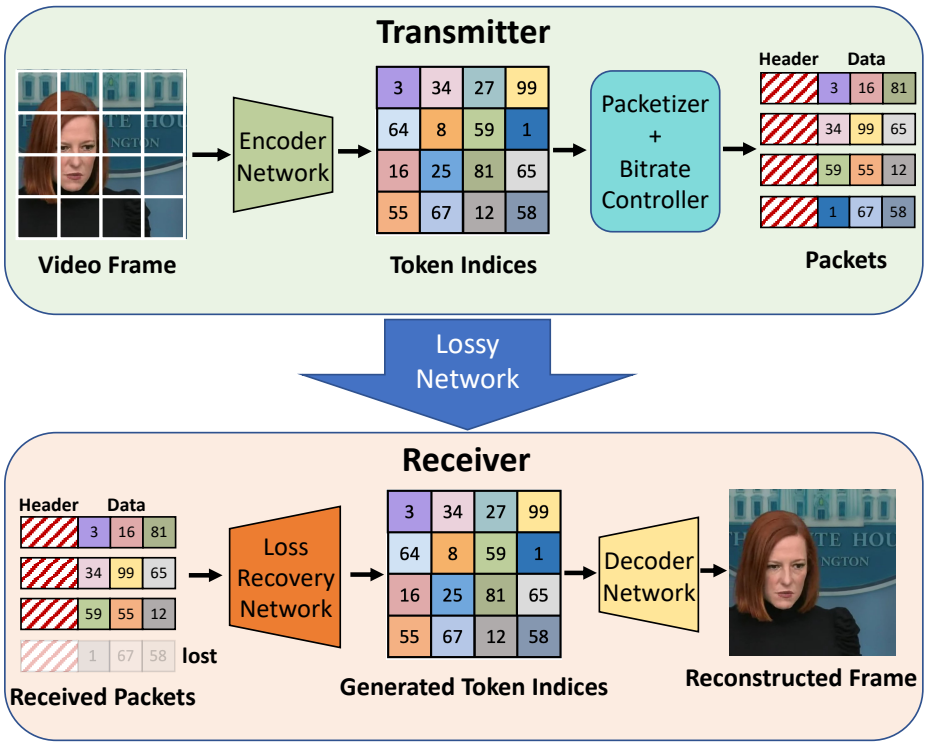


Figure 5-1: Overview of Reparo. An encoder-decoder pair converts between RGB image and quantized image tokens, while we introduce new modules to affect the packetization, bitrate, and loss recovery in the quantized token space to improve loss-resilience.

5.2.2 Reparo Components

5.2.2.1 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. Since the number of tokens in an image is much smaller than the number of pixels, this reduces the search space of generative models. Each token represents a vector in feature space. A set of tokens is selected for the codebook by training a neural network to identify a small number of feature vectors that can best generate all images in the training dataset.

We note 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. These tokens can simply be transmitted as indices without the need to transmit the actual tokens, providing significant

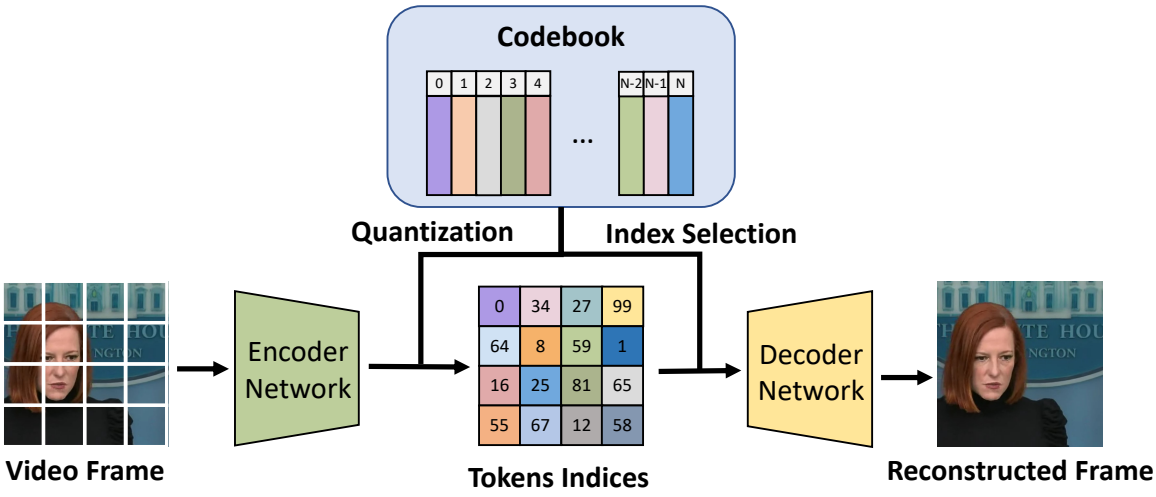


Figure 5-2: Token-based neural codec. The encoder converts patches from video frames into features and uses a codebook to quantize the features into tokens by finding the nearest neighbor of each feature in the codebook. The decoder then uses the tokens to reconstruct the video frame.

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

We use a particular tokenizer called VQGAN [53], which consists of an encoder, a decoder, and a codebook (see Fig. 5-2). 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. The number of tokens dictates the size of each patch within an image because the image is divided into patches, each mapped to a token. 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. 5-12.

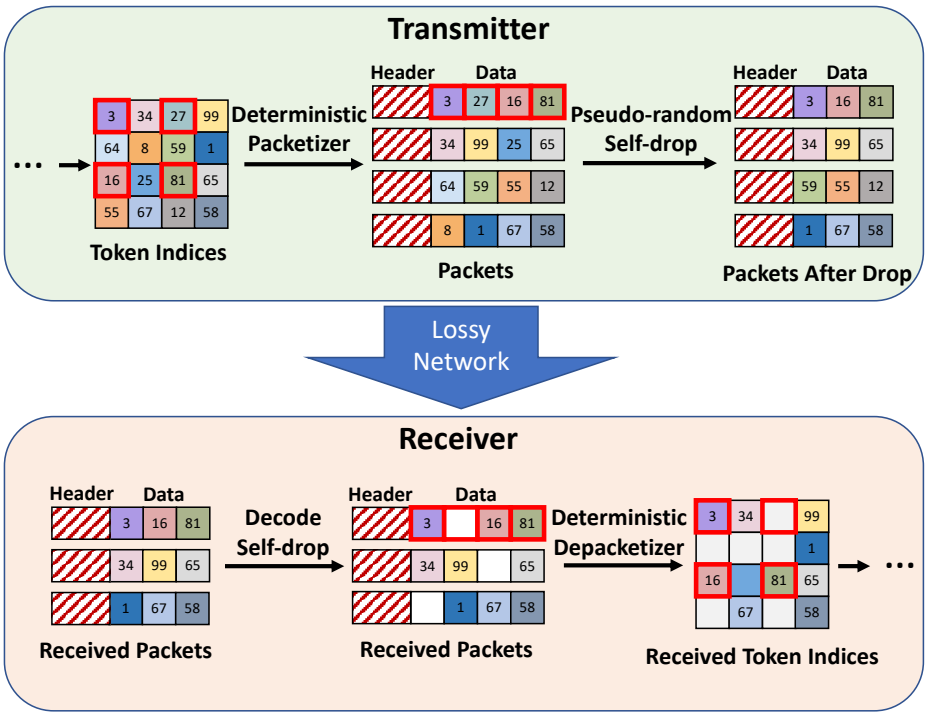


Figure 5-3: The transmitter first uses a deterministic packetizer to wrap image tokens into packets. Then a bitrate controller 'self-drops' some tokens in each packet to adapt to the target bitrate before even transmitting the tokens on the network and losing more tokens to packet drops. The receiver first decodes which tokens among the received packets are dropped by the bitrate controller. It then depacketizes those received packets to extract the received token indices. Any missing tokens (including those from lost packets) are identified and recovered using the loss recovery module.

5.2.2.2 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 image space are the most helpful for recovery when a token is lost and losing all adjacent tokens would be detrimental.

In Fig. 5-3, the first step in the green box labelled "Transmitter" shows the token wrapping strategy for a toy 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 strategy is just one of many options to wrap tokens into packets while avoiding placing adjacent tokens in the same packet. The chosen 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.

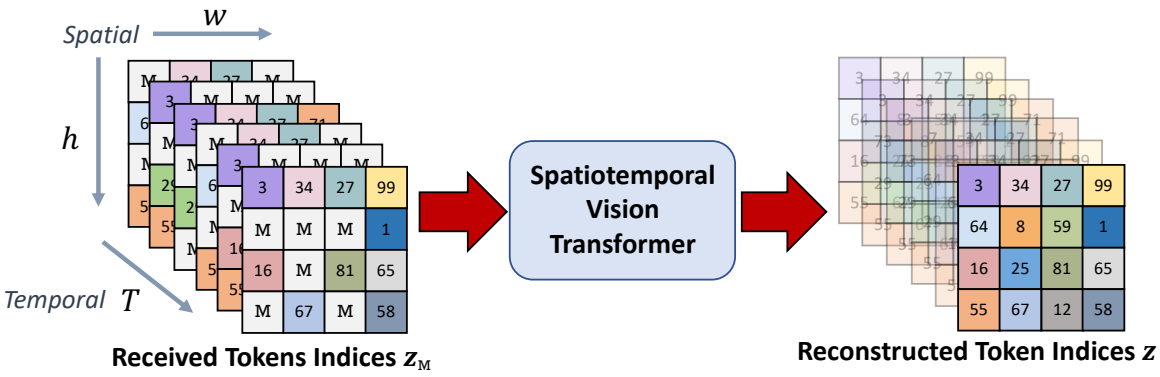


Figure 5-4: Loss recovery module. It uses a spatio-temporal vision transformer to generate any tokens that are lost using domain knowledge about human faces along with the received tokens in the last several frames.

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.

5.2.2.3 The Bitrate Controller

Video conferencing applications often need to adjust their bitrate in response to network congestion. In Gemino, this was achieved by altering the extent of compression and down-sampling to meet the desired bitrate. Traditional codecs alter the extent of quantization to control the 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 (and potentially losing more tokens via packet drops in the network). Remarkably, Reparo can tolerate up to 50% token loss with only a minimal impact on video PSNR, as demonstrated in Fig. 5-5. In practice, Reparo chooses the tokens it drops deterministically based on the frame index and packet index (Fig. 5-3 top row right). This is to ensure that the receiver can easily identify which token locations were self-dropped based simply on the frame and packet index in the received packet's header. With this information, the receiver can identify which tokens were lost by the bitrate controller (Fig. 5-3 bottom row left), and differentiate it from token locations lost to packet loss. The next subsection describes how the receiver decodes all the missing tokens collectively.

5.2.2.4 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 and torso based on a subset of those tokens. Similarly, it can produce the token corresponding to a moving arm 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. 5-4. 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) [51]. Transformers have gained widespread popularity in computer vision and natural language processing tasks for predicting missing image patches or words [50, 51, 64]. 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 [33]. 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 though Reparo uses tokens from previous frames for loss recovery, a loss of tokens or packets in prior frames does not cause Reparo to stall like traditional codecs. This is because the spatio-temporal ViT utilizes only the actually *received* tokens of the six previous frames while decoding the current frame. This allows reuse of received tokens across frames to achieve a better bitrate and loss rate but every new frame is generated and decoded regardless of the previous frame's generation result. If more tokens are lost in the previous frames, the quality of the synthesized current frame may be poorer, but Reparo will never stop generating or decoding, unlike classical codecs.

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 our custom 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. 5-4). This ensures a fixed input sequence length to the model 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). However, we only feed the last or current frame's tokens into the codec decoder to produce a synthesized frame. The loss recovery module is trained end-to-end 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. 5-1. 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,

$$\mathcal{L}_{reconstructive} = -\mathbb{E}_z\Big(\sum_{\forall i,j,k, m_{ijk}=1, k=T} \log\big(p(z_{ijk}|z_M)\big)\Big), \tag{5.1}$$

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 [64].

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 treat all un-received packets as lost. We can determine the exact locations of the missing tokens by placing the received tokens in their respective positions corresponding to a $h \times w$ grid of frame patches. We leverage all the tokens *received* from the previous 6 frames to perform spatio-temporal loss recovery for the missing tokens of the current frame .

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 is fed into the neural decoder to generate the RGB frame for display.

5.3 Evaluation

We evaluate Reparo using a version of an experimental video conferencing platform called Ringmaster [17] that has support for packet loss emulation and state-of-the-art FEC schemes. We describe our baselines and experimental setup in §5.3.1. We evaluate Reparo under network scenarios with random packet loss in §5.3.2, and under packet losses induced by a rate-limited bottleneck link in §5.3.3. We discuss Reparo's parameter choices and latency overheads in §5.3.4.

5.3.1 Experiment Setup

Baseline. The Ringmaster platform that we use for evaluation is equipped with Tambur [126], which is a recent streaming-codes based FEC solution atop the VP9 video codec [109] 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) [79], the CelebAHQ dataset (30,000 images) [78], and the TalkingHeads dataset (∼30 hours of video) [150]. These datasets comprise high-resolution human face images and videos, making them ideal for training Reparo, which is aimed at improving video conferencing quality. For training the loss recovery module, we exclusively

used the TalkingHeads dataset since it provides video clips (not just images) which the loss recovery module needs.

For evaluating Reparo and the baseline 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 center-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 [126], 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 [17] 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. We used the Simple DirectMedia Layer (SDL) library for recording displayed frames, and the PyTorch Image Quality (PIQ) library [80] for computing 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, 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 (§5.3.2), and (2) a rate-limited link (§5.3.3). 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 [126]. 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 mimic Tambur's evaluation, which used a large corpus of traces from Microsoft Teams [126] to compute approximate statistics. 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 6 KB and a drain rate of 320 Kbps.

Reparo Parameters. In our experiments, we use a 512×512 frame and compress it into 32×32 tokens. We use a codebook of size 1024 which means that 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 re-transmit it during video conferencing. Each frame's tokens are distributed across 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). Up to 50% of Reparo's tokens can be dropped to match the target bitrate using the "self-drop" mechanism described in §5.2.2.3. 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. 5-12.

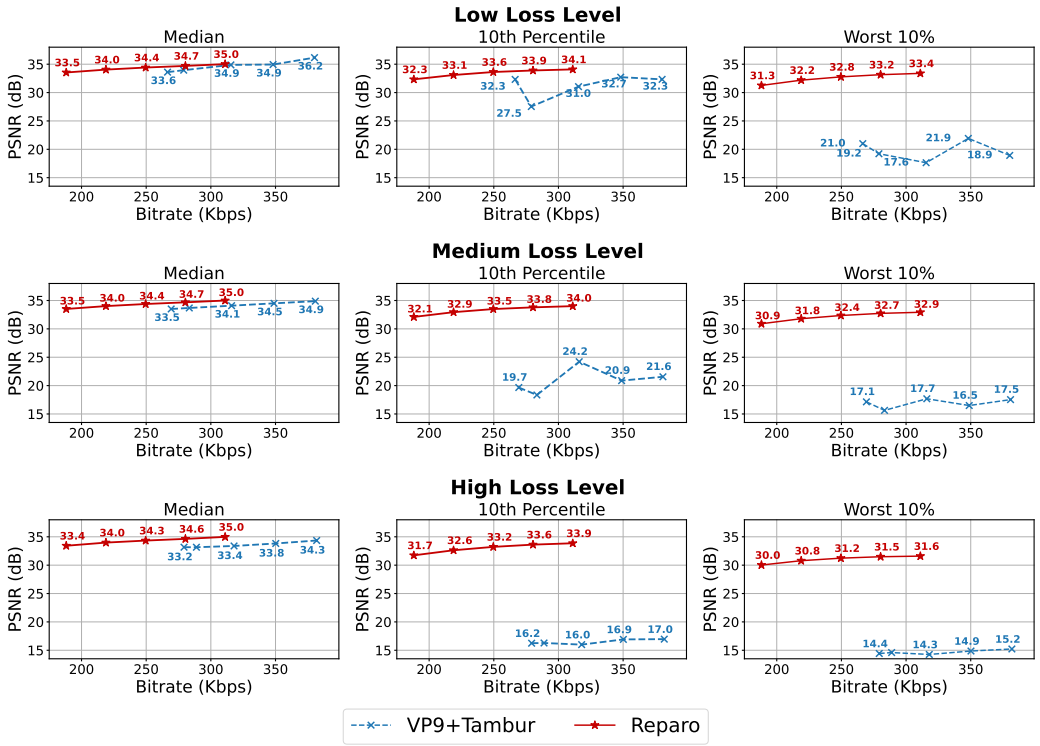


Figure 5-5: We report the median, 10th percentile, and worst 10% PSNR of VP9+Tambur and Reparo under different loss levels. We vary the target bitrate of Reparo and VP9+Tambur to cover different achieved bitrates. Reparo's visual quality at the tail is significantly better than VP9+Tambur across all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

5.3.2 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 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. 5-5, 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

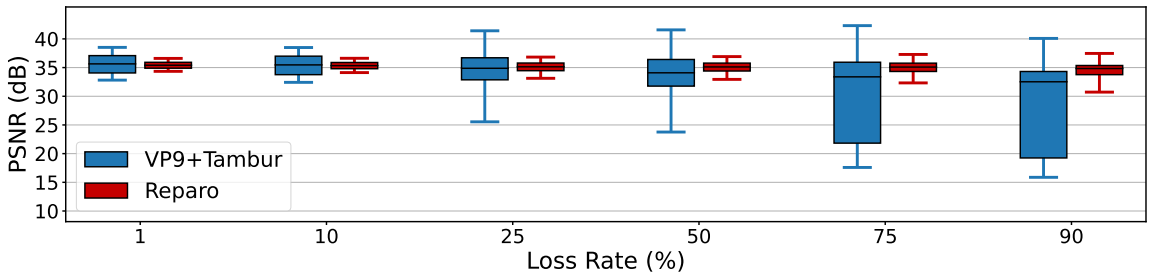


Figure 5-6: PSNR distribution across frames with Tambur and Reparo under different packet loss rates (for a bitrate of ∼320 Kbps). The box denotes the 25th and 75th percentile PSNR, the line inside the box denotes the median PSNR while the whiskers denote average PSNR ± 1.5×standard deviation. Reparo 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.

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.

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. 5-6, 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 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 stable under average network conditions, and more 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.

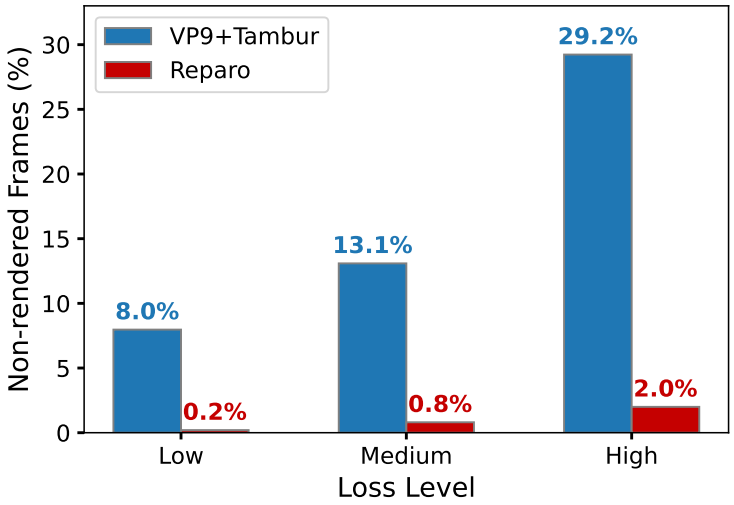


Figure 5-7: Comparison of percentage of non-rendered frames between Reparo and VP9+Tambur. VP9+Tambur has many more non-rendered frames than Reparo at all loss levels.

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+Tambur for low-quality rendered frames.

We evaluated Reparo and VP9+Tambur at similar bitrates (∼320 Kbps) under various loss levels. As shown in Fig. 5-7, 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. 5-8 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 time-frame 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

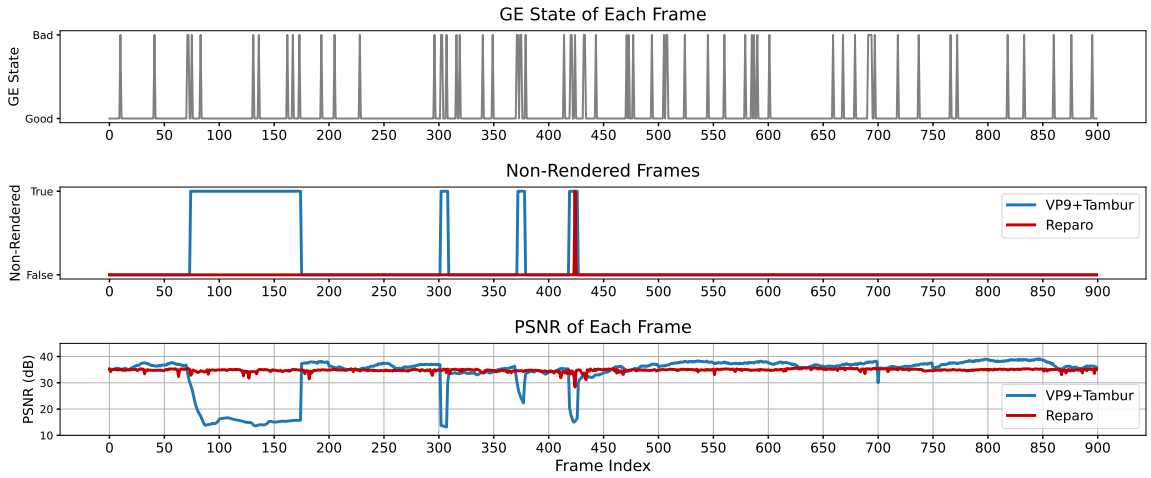


Figure 5-8: Time series comparing Tambur and Reparo on one video and loss pattern. VP9+Tambur experiences short freezes every time a set of frames are lost with a corresponding decrease in PSNR. Reparo continues rendering frames and its visual quality is a lot more stable throughout the interval.

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 time-frame 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. 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. 5-9. 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 3$^{\text{rd}}$, 4$^{\text{th}}$, and 5$^{\text{th}}$ frames are initially lost, followed by the loss of the 8$^{\text{th}}$ frame. VP9+Tambur does not render any frames between the 3$^{\text{rd}}$ and 10$^{\text{th}}$ 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

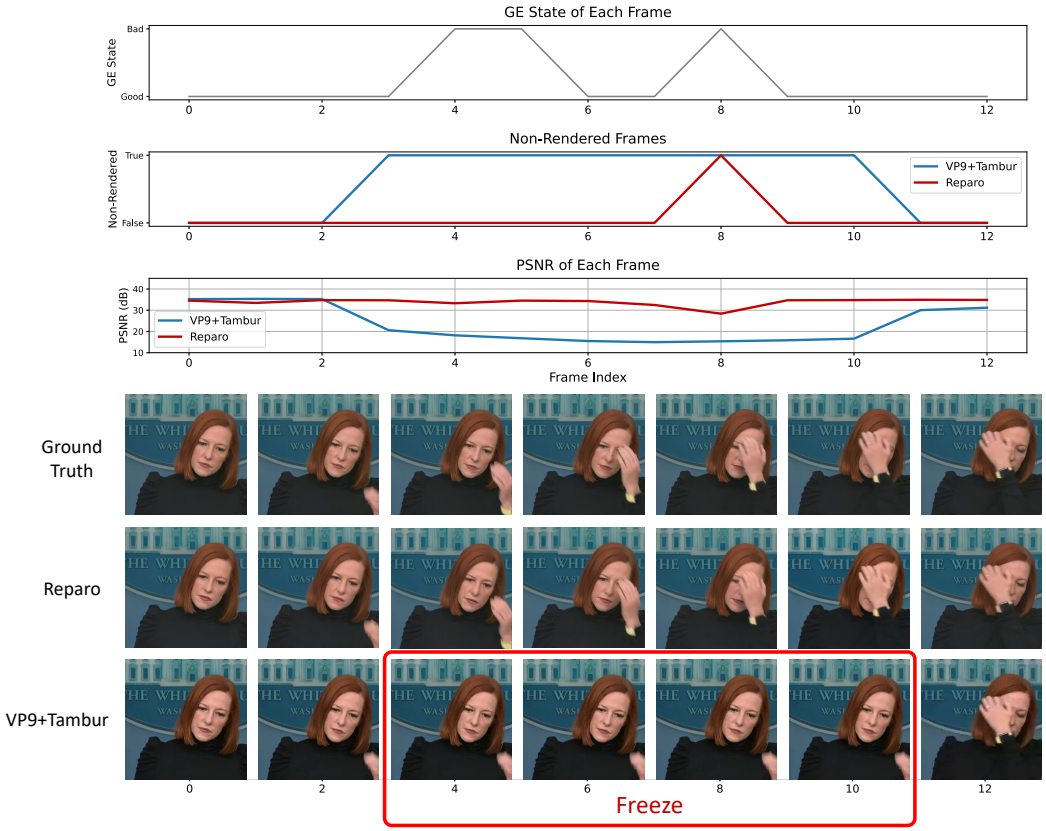


Figure 5-9: Qualitative results of VP9+Tambur and Reparo during a Tambur's short freeze of 8 frames. The GE loss channel is in a "bad" state at frames 4, 5, 6, and 8, causing packet losses for both VP9+Tambur and Reparo. VP9+Tambur completely freezes from frames 3 to 10 because of lost packets, leading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

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 continues to render frames throughout and 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 8^{th} frame, it rapidly recovers once later frames receive sufficient packets and tokens for high-quality generation.

5.3.3 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

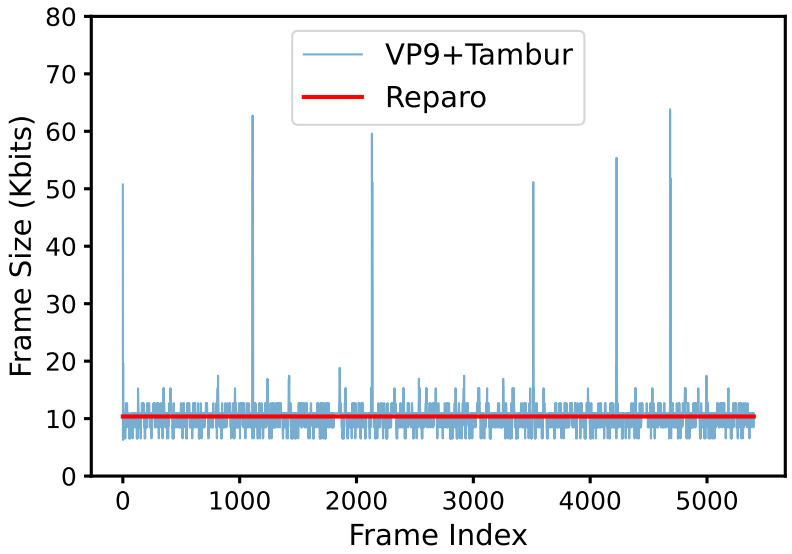


Figure 5-10: Per frame sizes of VP9+Tambur and Reparo for a 3 minute video. Reparo maintains the same frame size across all frames while VP9 shows variance both across adjacent predicted frames, and across periodic keyframes that are large.

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. 5-10, 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 [140]. 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. 5-11, 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

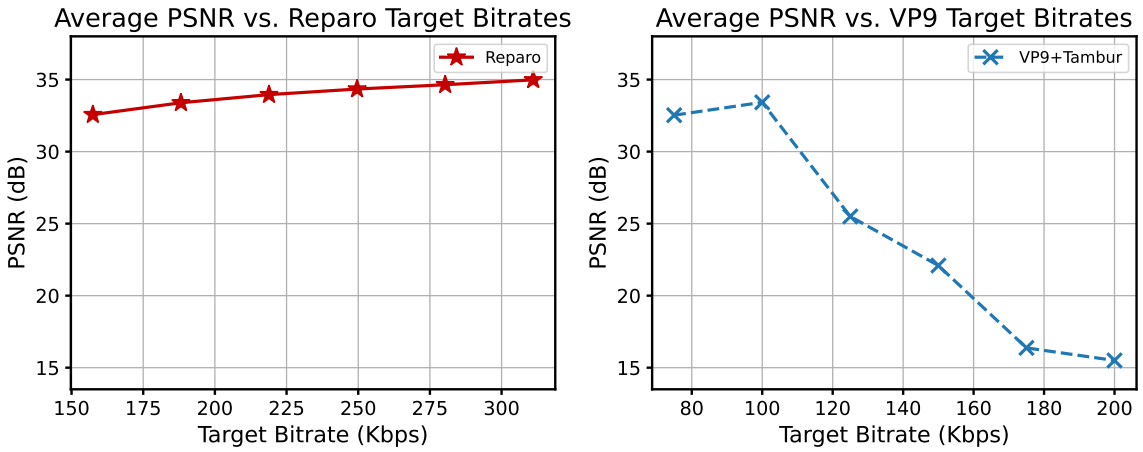


Figure 5-11: Average PSNR of Reparo and VP9+Tambur with different target bitrates for a fixed link capacity of 320 Kbps. Reparo's average PSNR improves as the target bitrate is increased. However, VP9+Tambur starts experiencing loss in its fixed-size queue beyond a target bitrate of 120 Kbps **due to large keyframes that do not fit in the queue.**

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. 5-11, 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 [42] 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 capac-

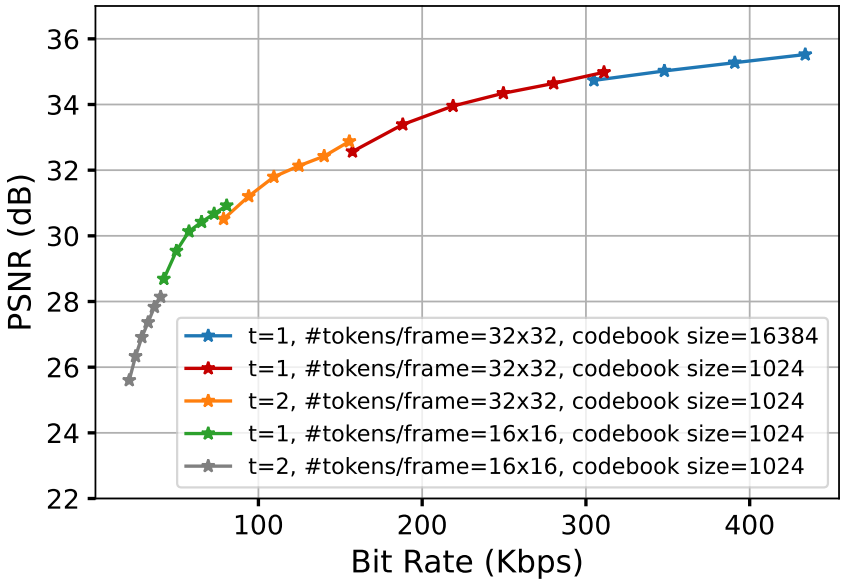


Figure 5-12: Variants of Reparo that operate in different bitrate regimes. Reparo achieves different bitrates by varying the number of tokens per frame, its codebook size, and the number of frames jointly encoded.

ity, and the best performance is achieved by setting the target bitrate near the link capacity.

5.3.4 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 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. 5-12, 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). Alternatively, Reparo can use larger patches and fewer tokens per frame to decrease the bitrate. However, this means that patches are less-specific and more coarse-grained, leading to worse quality (red curve and green 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.

	Encoder	Packetization	Loss Recovery	Decoder
Latency (ms)	9.2±0.1	0.5±0.009	35.6±1.2	15.5±0.2

Table 5-1: 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.

Latency. In Tab. 5-1, 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 [140]. We expect that standard techniques to improve model efficiency [68, 138, 52] can further reduce the latency.

5.4 Summary

This chapter describes Reparo, a novel loss-resilient generative video conferencing architecture that uses generative deep learning models to reconstruct missing information without sending redundant packets or using retransmissions. Instead, the receiver reconstructs missing parts of frames using its knowledge of how visual objects look and relate to each other. Our approach offers several advantages, including maintaining a stable and 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 codes for FEC. Reparo achieves similar PSNR as VP9+Tambur under mild loss levels and significantly improves over VP9+Tambur under heavy loss levels, while also nearly 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.

Chapter 6

Limitations

6.1 Limitations Common to All Three Solutions

This thesis proposes replacing traditional video codecs (*e.g.,* VP8, VP9, AV1) with neural networks that synthesize high-resolution video frames from compressed representations of the same. However, such techniques do have several limitations, particularly their computational complexity and their lack of generalizability.

Classical codecs are widely supported by a range of hardware from phones to expensive datacenter servers. This is because they have become so commonplace that support for these codecs is baked directly into each hardware type. In contrast, many existing video calling devices simply do not have the graphical processing units or GPUs necessary to support the kinds of operations neural networks require. Specifically, they cannot run the expensive matrix multiplication operations that most generative networks require in real-time or at the rate that video frames are sampled and rendered on a video conferencing call. We also acknowledge that many of the regions that suffer from poor video conferencing quality of experience (packet loss or low-bandwidth conditions) are also regions that have limited access to high-end devices that have good power and compute profiles. However, we believe that device improvements year on year are trending in a favorable direction, particularly with the emergence of neural engines for both Apple and Android devices [26, 5]. We also believe that the growing literature on model compression and optimization [60, 137, 41, 138] can be leveraged to improve the runtime and power consumption of all three models. In the meantime, an incremental deployment approach would involve running neural synthesis at edge GPUs close to the receiver. This allows receivers to benefit from better reconstruction quality as long as they have a reliable high-bandwidth pathway to the edge server and eliminates the need for a similar high-bandwidth reliable connectivity through the entire sender–receiver path or high-end devices with GPU capabilities.

Video compression techniques based on neural networks like the ones proposed in this

thesis also do not generalize to the same extent as classical codecs. Classical codecs are specified and implemented to work on all types of video regardless of their content. The same codecs are used to compress on-demand video with rapidly changing scenes as well as video conferencing videos that focus on individuals in largely stationary orientations through a call. However, Gemino and Reparo leverage the fact that video conferencing is restricted to a narrow distribution of talking-head videos to extract their compression gains. Such techniques need fallback mechanisms when out-of-distribution video frames (*e.g.,* a child in the background, a change of hair style, emojis and avatars) are encountered to ensure that the decoded video does not show any artifacts detrimental to the conferencing experience. Further, these compression strategies do not directly translate to on-demand or other video types; they need to be re-engineered for the specific task at hand.

6.2 Gemino Limitations

While Gemino greatly expands the operating regime for video conferencing to very low bitrates, it incurs significant overheads in the form of training costs for codec-in-the-loop training and personalization. It compresses better than VPX, but the encoding and decoding processes are quite a bit slower than VPX. Further, as mentioned above, Gemino is not as widely supported on devices without access to some graphical processing engine. Running NetAdapt [156] to perform layer-by-layer pruning is also quite expensive to do on a per-model basis. We alleviate some of these concerns by only training for the lowest bitrate that Gemino supports for a given resolution, and by running NetAdapt [156] once for a generic model to identify the architecture before fine-tuning it on a per-person basis. Yet, the training costs for such a high degree of content adaptation can be prohibitively expensive ($\sim$30 hours for one configuration on an A100 GPU).

In Chapter 7, we discuss some techniques to overcome the expensive training costs associated with Gemino's personalization and codec-in-the-loop training. In addition to that, we believe that NetAdapt can be adapted to our use case to identify an overarching optimized architecture that can then be carefully trained using transfer learning from its larger counterpart across a wide range of configurations and people. Further, NetAdapt [156] is only one technique amongst a large suite of model optimization approaches, including knowledge distillation [60], sparsification [57, 55], quantization [62] and broader neural architecture search approaches that optimize the entire model [41, 138]. We believe that with more targeted optimizations for particular devices, we can do even better. Such optimizations become more salient when operating on higher-resolution video (*e.g.,* 4K, UltraHD) and in higher bandwidth regimes ($\sim$ 5 Mbps). We leave an exploration of such optimizations to future work.

Gemino, though trained on random pairs of reference and target frames, always uses

the first frame of the test video as its reference frame. The reconstruction fidelity can be improved by using reference frames close to each target frame. However, merely sending more frequent reference frames periodically incurs very high bitrate costs due to their high resolution. We leave to future work a more thorough investigation of reference frame selection mechanisms that weigh these tradeoffs to squeeze the maximum accuracy for a given compression level even with *a single reference frame*. Chapter 4 approaches this issue differently: leveraging multiple reference frames using attention instead of carefully selecting a single good reference frame.

6.3 Gemino (Attention) Limitations

Though Gemino (Attention) uses more than one reference frame unlike Gemino, the solution comes at significant compute costs. Specifically, the attention-based architecture involves many more matrix multiplication operations on large multi-dimensional matrices. This results in an increase in FLOPs and consequently, inference time as the number of references is increased. However, unlike Gemino, its attention counterpart has not been optimized to use depthwise-convolutions, knowledge distillation or minimal number of kernels and filters for optimal reconstruction. These standard model compression techniques can be applied to Gemino (Attention) as well to improve its reconstruction time across the hardware spectrum. Recent developments such as FlashAttention [48] can further be employed to align the attention mechanism with IO bottlenecks in a way that reduces memory overheads. Additionally, methods developed for alternate transformer models [147, 87, 125] can be leveraged to compress the key-value pairs calculated from reference frames to perform attention over smaller dimensions or to create approximations of the attention mechanism altogether. Given that reference frames tend to remain fixed throughout the entire inference process and often contain substantial redundancy, both compression and approximation techniques that leverage the redundancy can greatly reduce the computational costs.

6.4 Reparo 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 [51, 33], arguably more than even Gemino [134]. As a result, it requires *multiple* 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 is very far from being suitable for low-end devices such as smartphones or tablets. However, as seen with Gemino, machine learning models can typically be sped up for edge device deployment using more efficient

model architectures [68, 138, 52, 118], hardware design [57, 61, 164], and techniques such as knowledge distillation [66]. We leave an investigation of such optimizations, specifically targeted at transformer architectures, 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. Third, Reparo's performance during low-loss scenarios is worse than existing codecs and state-of-the-art solutions for packet loss. We believe that recent developments [161, 160, 90] in token-based architectures can close this gap and also outperform existing solutions.

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.

Chapter 7

Conclusion

7.1 Thesis Summary

This thesis proposes two neural codecs for improving video conferencing under low-bandwidth and lossy network conditions. Together, these codecs point towards a future with smoother and better video conferencing experience regardless of the underlying network.

First, we develop Gemino, a neural codec that upsamples low-resolution frames compressed at audio-like bitrates to its full resolution using a high-frequency conditional super-resolution pipeline. Gemino achieves comparable video quality to VP8 and VP9 with one-fifth and one-half of the bitrate respectively. It runs in real-time on a Titan X GPU at 1024×1024 resolution, and runs in 90 ms on a Jetson TX2 system. Next, we improve upon Gemino's design by replacing its optical flow with an attention module instead. The revamped design, Gemino (Attention), combines information across multiple reference frames to further improve the synthesized frame's quality. While this technique shows limited improvements with highly curated datasets, the visual quality of the reconstruction improved by nearly 1 dB for both SSIM and PSNR with ten reference frames on a more diverse dataset.

Lastly, we design Reparo, a token-based generative codec that nearly eliminates frame freezes or non-rendered frames during a video call. Reparo does this by learning a codebook or visual dictionary that allows it to reconstruct small patches of an image that, when combined appropriately, form the final rendered image. This codec compresses very efficiently because it simply transmits token (patch) indices, and can synthesize any missing tokens by exploiting the correlations across token locations. Such an approach improves the reconstruction quality by nearly 18 dB in PSNR under high-loss conditions.

In the rest of this chapter, we describe some of the open questions left by this thesis, and envision a future where neural video conferencing is truly ubiquitous.

7.2 Future Work

Feed-forward Networks for Content Adaptation. In Chapter 3, we show that finetuning models on a per-person and per-codec-configuration basis improves Gemino's reconstruction fidelity allowing it to support high-quality video conferencing at audio-like bitrates. However, this comes at a significant compute cost. Specifically, each finetuning task for a particular person and bitrate task takes nearly 30 hours even on an A100 GPU. While this can be sped up to an extent via distributed training, this is still prohibitively expensive. We believe that one way to alleviate this is to design entirely *feed-forward* networks for content adaptation. Such a design would involve a common model architecture based on the task at hand. The weights of the common model would be computed by a hyper-network which takes the specific content at hand and simply sets the weights of the common model. In the context of Gemino's high-frequency conditional super-resolution pipeline, the architecture described in Fig. 3-2 in §5.2 would remain unchanged. However, based on the person and the codec settings, a hyper-network would determine what weights for each layer in Fig. 3-2 results in the highest reconstruction accuracy. This allows us to train *exactly once* a single pair of models: the core model and the weight-setting hyper-network, but use them at inference time multiple times on different content distributions to still extract just as good quality as finetuning the core model repeatedly for each content distribution. It also makes it easier to compute model updates for model streaming use cases such as in SRVC [83] and AMS [82]. We anticipate that some of the optimization benefits from running NetAdapt [156] for Gemino can also be extended to such a feed-forward design too by merely retaining weights produced by the hyper-network for those layers that remain after the pruning process.

Content Adaptation for Bias Reduction. A benefit of content-specific models is that they can trained to reasonable accuracy on a generic corpus before finetuning it on a small dataset consisting of a restricted distribution. The resulting model has high reconstruction accuracy on the specific distribution it was finetuned on, as seen with Gemino's personalized models (§3.2.3). We believe that this can be used to reduce bias typically associated with models trained on datasets that are not representative of the distributions that they are tested on [39, 38, 4]. A commonly cited reason for why bias exists is that data is hard to collect on specific minority populations [74, 58]. However, our approach suggests that finetuning on a small curated dataset on a restricted distribution can recover any accuracy that a generic model loses out on. Building on this insight, we anticipate that a collection of such content-adapted models each trained on smaller minority populations can be used to improve the robustness of pipelines that use trained models for highly sensitive tasks. The idea here is similar to ensembling or bagging [127] where the ultimate output is an

average or minimum of all individual content-adapted model outputs.

Person-specific Feature Corpuses. Chapter 4 proposes an attention based design for high-frequency-conditional super-resolution that leverages multiple reference frames. All the reference frames are encoded to produce a series of key-value feature pairs at multiple resolutions. These features are then attended to based on query features obtained from the low-resolution target frame meant to be upsampled to its fullest resolution. However, the evaluation in §4.3 merely concatenates the features obtained from all of the reference frames. This leaves the attention module maximum flexibility to combine information across all reference frames however it deems most useful for the final reconstruction quality. But, this has the unfortunate side effect of inducing high compute overheads. A more efficient strategy is to aggregate information across all the key-value feature pairs obtained from all reference frames into a compact summary that can still improve reconstruction quality without as much compute. Such aggregation can range from simple clustering mechanisms that remove redundant features common to multiple reference frames to more complex learnt approaches. This summarization can be used to generate person-specific feature corpuses that can then be leveraged for downstream tasks without needing to retrain the weights of the entire larger model for that particular task. This poses an alternate way to perform content-adaptation that may not be entirely feed-forward, but still reduces the overheads from retraining the entire model for every distribution to obtaining a small summary per distribution.

Loss-resilient Low-Bandwidth Video Conferencing. This thesis describes two separate techniques that independently target low bandwidth and loss-resilient video conferencing. Gemino [134, 135] lower the bandwidth use by sending low-resolution video over the Internet but neurally upsampling it to the desired resolution. Reparo [92] develops a token-based generative codec that does not use any temporal dependency across frames, allowing reconstruction for any frame independent of the outcome of prior frames' decoding process. Combining both of these approaches will result in a codec that is both loss-resilient as well as extremely bandwidth efficient. However, we anticipate that the compute costs associated with such a design will be significantly higher than either individual approach. We leave to future work an exploration of techniques to efficiently combine both techniques. A more responsive transport layer such as the one in Vidaptive [77] will further improve the system's ability to detect loss and poor network conditions, allowing it to use the approaches in Gemino and Reparo more appropriately while minimizing the compute overheads during good network conditions.

7.3 Making Neural Video Conferencing Ubiquitous

This thesis shows the immense benefits that generative methods provide to video conferencing under poor network conditions. However, many of the regions that most need better video conferencing experience do not have wealthy populations that can afford high-end mobile phones or laptops that can support neural computations. We take the optimistic outlook that given the pace of hardware improvements over the last decade, devices will soon be able to support some form of basic neural methods if not compute-intensive ones like transformer architectures. We believe that simple super-resolution approaches without reference frames might be a good starting deployment strategy that shows some of the benefits that neural networks and machine learning afford to video conferencing. As device capabilities improve, we anticipate deploying increasingly complex approaches. In the meantime, we expect that baseline network conditions across the world will also improve and expand the realm of video conferencing to higher quality experiences such as 4K and UltraHD video calls, as well as immersive experiences such as AR/VR. This thesis paves the way for democratizing access to such experiences as and when they develop even if the network conditions across the world cannot instantly support these more bandwidth-intensive future video experiences.

Bibliography

[1] 20 Video Conferencing Stats: What it Means for Your Business. https://www.pepperlandmarketing.com/blog/video-conference-stats.

[2] A Measurement Tool for Videoconferencing User Experience. https://dspace.mit.edu/bitstream/handle/1721.1/151610/jin-cljin-meng-eecs-2023-thesis.pdf.

[3] aiortc. https://github.com/aiortc/aiortc.

[4] Amazon scraps secret AI recruiting tool that showed bias against women. https://dspace.mit.edu/bitstream/handle/1721.1/151610/jin-cljin-meng-eecs-2023-thesis.pdf.

[5] Android Neural Networks API. https://source.android.com/docs/core/ota/modular-system/nnapi.

[6] Apple Vision Pro. https://www.apple.com/apple-vision-pro/.

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

[8] Change the quality of your video. https://support.google.com/youtube/answer/91449?hl=en.

[9] Explore hardware options to enable Zoom. https://explore.zoom.us/en/workspaces/conference-room/.

[10] Google Meet Hardware. "https://workspace.google.com/products/meet-hardware/".

[11] ISO IEC 11172-2 1993. https://www.iso.org/obp/ui/en/#iso:std:iso-iec:11172:-2:ed-1:v1:en.

[12] Keras Attention Layer. https://keras.io/api/layers/attention_layers/attention/.

[13] Median Country Speeds March 2023. https://www.speedtest.net/global-index.

[14] NVIDIA TensorRT. https://developer.nvidia.com/tensorrt.

[15] Opus interactive audio codec. https://opus-codec.org/.

[16] Pyav documentation. https://pyav.org/docs/stable/.

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

[18] Steve Jobs unveiled the first iPhone 16 years ago — look how primitive it seems today. https://www.businessinsider.com/first-phone-anniversary-2016-12.

[19] The MPEG Home Page. https://web.archive.org/web/20080708183405/http://mpeg.chiariglione.org/achievements.htm.

[20] The state of video conferencing 2022. https://www.dialpad.com/blog/video-conferencing-report/.

[21] The Very First Webcam Was Invented to Keep an Eye on a Coffee Pot at Cambridge University. https://www.openculture.com/2021/09/the-very-first-webcam-was-invented-to-keep-an-eye-on-a-coffee-pot-at-cambridge-university.html.

[22] Top Tips To Improve Audio Quality in Your Zoom Meetings. https://uit.stanford.edu/news/top-tips-improve-audio-quality-your-zoom-meetings.

[23] Videoconferencing app usage 'hits 21 times pre-Covid levels'. https://www.avinteractive.com/news/collaboration/usage-mobile-video-conferencing-apps-including-zoom-grew-150-first-half-2021-05-08-2021/.

[24] VP8 Chromium Implementation. https://chromium.googlesource.com/external/webrtc/+/143cec1cc68b9ba44f3ef4467f1422704f2395f0/webrtc/modules/video_coding/codecs/vp8/vp8_impl.cc.

[25] WebRTC connectivity. https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API/Connectivity.

[26] What Is Apple's Neural Engine and How Does It Work? https://www.makeuseof.com/what-is-a-neural-engine-how-does-it-work/.

[27] What is the Best Audio Codec for Online Video Streaming? https://www.dacast.com/blog/best-audio-codec/.

[28] Worldwide broadband speed league 2022. https://www.cable.co.uk/broadband/speed/worldwide-speed-league/#speed.

[29] Zoom System Requirements. https://support.zoom.us/hc/en-us/articles/201362023-Zoom-system-requirements-Windows-macOS-Linux.

[30] Project Starline: Feel like you're there, together. https://blog.google/technology/research/project-starline/, 2021.

[31] Unlimited HD Video Calls. `https://trueconf.com/features/modes/videocall.html`, 2021.

[32] N. Ahmed, T. Natarajan, and K.R. Rao. Discrete cosine transform. *IEEE Transactions on Computers*, C-23(1):90–93, 1974.

[33] 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.

[34] 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.

[35] 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.

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

[37] Gedas Bertasius, Heng Wang, and Lorenzo Torresani. Is space-time attention all you need for video understanding? In *ICML*, volume 2, page 4, 2021.

[38] Tolga Bolukbasi, Kai-Wei Chang, James Zou, Venkatesh Saligrama, and Adam Kalai. Man is to computer programmer as woman is to homemaker? debiasing word embeddings, 2016.

[39] Joy Buolamwini and Timnit Gebru. Gender shades: Intersectional accuracy disparities in commercial gender classification. In *Conference on fairness, accountability and transparency*, pages 77–91. PMLR, 2018.

[40] Jose Caballero, Christian Ledig, Andrew Aitken, Alejandro Acosta, Johannes Totz, Zehan Wang, and Wenzhe Shi. Real-time video super-resolution with spatio-temporal networks and motion compensation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4778–4787, 2017.

[41] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. *arXiv preprint arXiv:1908.09791*, 2019.

[42] 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.

[43] 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.

[44] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Meghan Cowan, Haichen Shen, Leyuan Wang, Yuwei Hu, Luis Ceze, et al. TVM: An automated end-to-end optimizing compiler for deep learning. *arXiv preprint arXiv:1802.04799*, 2018.

[45] Yu Chen, Ying Tai, Xiaoming Liu, Chunhua Shen, and Jian Yang. Fsrnet: End-to-end learning face super-resolution with facial priors. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2492–2501, 2018.

[46] Yue Chen, Debargha Murherjee, Jingning Han, Adrian Grange, Yaowu Xu, Zoe Liu, Sarah Parker, Cheng Chen, Hui Su, Urvang Joshi, et al. An overview of core coding tools in the AV1 video codec. In *2018 Picture Coding Symposium (PCS)*, pages 41–45. IEEE, 2018.

[47] Joon Son Chung, Arsha Nagrani, and Andrew Zisserman. Voxceleb2: Deep speaker recognition. *arXiv preprint arXiv:1806.05622*, 2018.

[48] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness, 2022.

[49] 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.

[50] 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.

[51] 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. *arXiv preprint arXiv:2010.11929*, 2020.

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

[53] 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.

[54] Sadjad Fouladi, John Emmons, Emre Orbay, Catherine Wu, Riad S Wahby, and Keith Winstein. Salsify: Low-latency network video through tighter integration between a video codec and a transport protocol. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 267–282, 2018.

[55] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

[56] Kunihiko Fukushima and Sei Miyake. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*, pages 267–285. Springer, 1982.

[57] 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.

[58] Timnit Gebru, Jamie Morgenstern, Briana Vecchione, Jennifer Wortman Vaughan, Hanna Wallach, Hal Daumé Iii, and Kate Crawford. Datasheets for datasets. *Communications of the ACM*, 64(12):86–92, 2021.

[59] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[60] Jianping Gou, Baosheng Yu, Stephen J Maybank, and Dacheng Tao. Knowledge distillation: A survey. *International Journal of Computer Vision*, 129:1789–1819, 2021.

[61] 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.

[62] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[63] Naomi Harte and Eoin Gillen. Tcd-timit: An audio-visual corpus of continuous speech. *IEEE Transactions on Multimedia*, 17(5):603–615, 2015.

[64] 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.

[65] 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.

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

[67] 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.

[68] 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.

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

[70] Zheng Hui, Xinbo Gao, Yunchu Yang, and Xiumei Wang. Lightweight image super-resolution with information multi-distillation network. In *Proceedings of the 27th acm international conference on multimedia*, pages 2024–2032, 2019.

[71] Zheng Hui, Xiumei Wang, and Xinbo Gao. Fast and accurate single image super-resolution via information distillation network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 723–731, 2018.

[72] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France, 07–09 Jul 2015. PMLR.

[73] Hao Jiang and Constantinos Dovrolis. Why is the internet traffic bursty in short time scales? 33(1):241–252, jun 2005.

[74] Eun Seo Jo and Timnit Gebru. Lessons from archives: Strategies for collecting sociocultural data in machine learning. In *Proceedings of the 2020 conference on fairness, accountability, and transparency*, pages 306–316, 2020.

[75] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European conference on computer vision*, pages 694–711. Springer, 2016.

[76] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European conference on computer vision*, pages 694–711. Springer, 2016.

[77] Pantea Karimi, Sadjad Fouladi, Vibhaalakshmi Sivaraman, and Mohammad Alizadeh. Vidaptive: Efficient and responsive rate control for real-time video on variable networks, 2023.

[78] 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.

[79] 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.

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

[81] Robert Keys. Cubic convolution interpolation for digital image processing. *IEEE transactions on acoustics, speech, and signal processing*, 29(6):1153–1160, 1981.

[82] Mehrdad Khani, Pouya Hamadanian, Arash Nasr-Esfahany, and Mohammad Alizadeh. Real-time video inference on edge devices via adaptive model streaming. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 4572–4582, October 2021.

[83] 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.

[84] 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.

[85] Jiwon Kim, Jung Kwon Lee, and Kyoung Mu Lee. Accurate image super-resolution using very deep convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1646–1654, 2016.

[86] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[87] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *ICLR*, 2020.

[88] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[89] 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.

[90] Doyup Lee, Chiheon Kim, Saehoon Kim, Minsu Cho, and Wook-Shin Han. Autoregressive image generation using residual quantization, 2022.

[91] 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.

[92] Tianhong Li, Vibhaalakshmi Sivaraman, Lijie Fan, Mohammad Alizadeh, and Dina Katabi. Reparo: Loss-resilient generative codec for video conferencing, 2023.

[93] Jingyun Liang, Jiezhang Cao, Yuchen Fan, Kai Zhang, Rakesh Ranjan, Yawei Li, Radu Timofte, and Luc Van Gool. Vrt: A video restoration transformer. *arXiv preprint arXiv:2201.12288*, 2022.

[94] Jingyun Liang, Jiezhang Cao, Guolei Sun, Kai Zhang, Luc Van Gool, and Radu Timofte. Swinir: Image restoration using swin transformer. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1833–1844, 2021.

[95] Jingyun Liang, Jiezhang Cao, Guolei Sun, Kai Zhang, Luc Van Gool, and Radu Timofte. Swinir: Image restoration using swin transformer. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1833–1844, 2021.

[96] Yi J Liang, John G Apostolopoulos, and Bernd Girod. Analysis of packet loss for compressed video: Effect of burst losses and correlation between error frames. *IEEE Transactions on Circuits and Systems for Video Technology*, 18(7):861–874, 2008.

[97] Y.J. Liang, J.G. Apostolopoulos, and B. Girod. Analysis of packet loss for compressed video: does burst-length matter? In *2003 IEEE International Conference on Acoustics, Speech, and Signal Processing, 2003. Proceedings. (ICASSP '03).*, volume 5, pages V–684, 2003.

[98] Bee Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, and Kyoung Mu Lee. Enhanced deep residual networks for single image super-resolution. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 136–144, 2017.

[99] 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.

[100] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 10012–10022, 2021.

[101] 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.

[102] Zhisheng Lu, Juncheng Li, Hong Liu, Chaoyan Huang, Linlin Zhang, and Tieyong Zeng. Transformer for single image super-resolution. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 457–466, 2022.

[103] Cheng Ma, Zhenyu Jiang, Yongming Rao, Jiwen Lu, and Jie Zhou. Deep face super-resolution with iterative collaboration between attentive recovery and landmark estimation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5569–5578, 2020.

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

[105] Arun Mallya, Ting-Chun Wang, and Ming-Yu Liu. Implicit warping for animation with image sets, 2022.

[106] Fabian Mentzer, George Toderici, David Minnen, Sung-Jin Hwang, Sergi Caelles, Mario Lucic, and Eirikur Agustsson. Vct: A video compression transformer. *arXiv preprint arXiv:2206.07307*, 2022.

[107] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Commun. ACM*, 65(1):99–106, dec 2021.

[108] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957*, 2018.

[109] 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.

[110] 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.

[111] Arsha Nagrani, Joon Son Chung, and Andrew Zisserman. Voxceleb: a large-scale speaker identification dataset. *arXiv preprint arXiv:1706.08612*, 2017.

[112] Vikram Nathan, Vibhaalakshmi Sivaraman, Ravichandra Addanki, Mehrdad Khani, Prateesh Goyal, and Mohammad Alizadeh. End-to-end transport for video qoe fairness. In *Proceedings of the ACM Special Interest Group on Data Communication*, pages 408–423. 2019.

[113] Maxime Oquab, Pierre Stock, Daniel Haziza, Tao Xu, Peizhao Zhang, Onur Celebi, Yana Hasson, Patrick Labatut, Bobo Bose-Kolanu, Thibault Peyronel, et al. Low bandwidth video-chat compression using deep generative models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2388–2397, 2021.

[114] Maxime Oquab, Pierre Stock, Daniel Haziza, Tao Xu, Peizhao Zhang, Onur Celebi, Yana Hasson, Patrick Labatut, Bobo Bose-Kolanu, Thibault Peyronel, et al. Low bandwidth video-chat compression using deep generative models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2388–2397, 2021.

[115] Eunbyung Park, Jimei Yang, Ersin Yumer, Duygu Ceylan, and Alexander C. Berg. Transformation-grounded image generation network for novel 3d view synthesis. In

Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), July 2017.

[116] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[117] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents, 2022.

[118] 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.

[119] 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.

[120] 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.

[121] Yurui Ren, Ge Li, Yuanqi Chen, Thomas H. Li, and Shan Liu. Pirenderer: Controllable portrait image generation via semantic neural rendering. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 13759–13768, October 2021.

[122] 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.

[123] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models, 2022.

[124] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[125] Aurko Roy, Mohammad Saffar, Ashish Vaswani, and David Grangier. Efficient content-based sparse attention with routing transformers. In *TACL*, 2020.

[126] 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.

[127] Omer Sagi and Lior Rokach. Ensemble learning: A survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 8(4):e1249, 2018.

[128] Henning Schulzrinne, Stephen Casner, Ron Frederick, and Van Jacobson. Rtp: A transport protocol for real-time applications, 1996.

[129] 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.

[130] 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.

[131] Aliaksandr Siarohin, Stéphane Lathuilière, Sergey Tulyakov, Elisa Ricci, and Nicu Sebe. First order motion model for image animation. *Advances in Neural Information Processing Systems*, 32, 2019.

[132] Aliaksandr Siarohin, Stéphane Lathuilière, Sergey Tulyakov, Elisa Ricci, and Nicu Sebe. First order motion model for image animation. In *Conference on Neural Information Processing Systems (NeurIPS)*, December 2019.

[133] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[134] Vibhaalakshmi Sivaraman, Pantea Karimi, Vedantha Venkatapathy, Mehrdad Khani, Sadjad Fouladi, Mohammad Alizadeh, Frédo Durand, and Vivienne Sze. Gemino: Practical and robust neural compression for video conferencing. In *To be presented as part of the 21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, 2024.

[135] Vibhaalakshmi Sivaraman, Xuan Luo, Anne Menini, Mohammad Alizadeh, and Rahul Garg. Multi-Resolution Multi-Reference Talking Head Synthesis via Implicit Warping. Technical report, Massachusetts Institute of Technology, Nov 2023.

[136] 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.

[137] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S Emer. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12):2295–2329, 2017.

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

[139] Marc Comino Trinidad, Ricardo Martin Brualla, Florian Kainz, and Janne Kontkanen. Multi-view image fusion. 2019.

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

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

[142] 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.

[143] Anna Volokitin, Stefan Brugger, Ali Benlalah, Sebastian Martin, Brian Amberg, and Michael Tschannen. Neural face video compression using multiple views, 2022.

[144] Anna Volokitin, Stefan Brugger, Ali Benlalah, Sebastian Martin, Brian Amberg, and Michael Tschannen. Neural face video compression using multiple views, 2022.

[145] Mythili Vutukuru, Hari Balakrishnan, and Kyle Jamieson. Cross-layer wireless bit rate adaptation. 39(4):3–14, aug 2009.

[146] A. Vyas, A. Katharopoulos, and F. Fleuret. Fast transformers with clustered attention. *arXiv preprint arXiv:2007.04825*, 2020.

[147] Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity, 2020.

[148] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Guilin Liu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. Video-to-video synthesis. *arXiv preprint arXiv:1808.06601*, 2018.

[149] 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.

[150] 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.

[151] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[152] WebRTC. `https://webrtc.org/`.

[153] Diana Wofk, Fangchang Ma, Tien-Ju Yang, Sertac Karaman, and Vivienne Sze. FastDepth: Fast Monocular Depth Estimation on Embedded Systems. *2019 International Conference on Robotics and Automation (ICRA)*, pages 6101–6108, 2019.

[154] Xiaoyu Xiang, Jon Morton, Fitsum A Reda, Lucas D Young, Federico Perazzi, Rakesh Ranjan, Amit Kumar, Andrea Colaco, and Jan P Allebach. Hime: Efficient headshot image super-resolution with multiple exemplars. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 1694–1704, 2023.

[155] Fuzhi Yang, Huan Yang, Jianlong Fu, Hongtao Lu, and Baining Guo. Learning texture transformer network for image super-resolution. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5791–5800, 2020.

[156] Tien-Ju Yang, Andrew Howard, Bo Chen, Xiao Zhang, Alec Go, Mark Sandler, Vivienne Sze, and Hartwig Adam. NetAdapt: Platform-Aware Neural Network Adaptation for Mobile Applications. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

[157] Hyunho Yeo, Chan Ju Chong, Youngmok Jung, Juncheol Ye, and Dongsu Han. Nemo: Enabling neural-enhanced video streaming on commodity mobile devices. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, MobiCom '20, New York, NY, USA, 2020. Association for Computing Machinery.

[158] 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.

[159] 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.

[160] Lijun Yu, Yong Cheng, Kihyuk Sohn, José Lezama, Han Zhang, Huiwen Chang, Alexander G. Hauptmann, Ming-Hsuan Yang, Yuan Hao, Irfan Essa, and Lu Jiang. Magvit: Masked generative video transformer, 2023.

[161] Lijun Yu, José Lezama, Nitesh B. Gundavarapu, Luca Versari, Kihyuk Sohn, David Minnen, Yong Cheng, Agrim Gupta, Xiuye Gu, Alexander G. Hauptmann, Boqing Gong, Ming-Hsuan Yang, Irfan Essa, David A. Ross, and Lu Jiang. Language model beats diffusion – tokenizer is key to visual generation, 2023.

[162] Egor Zakharov, Aleksei Ivakhnenko, Aliaksandra Shysheya, and Victor Lempitsky. Fast bi-layer neural synthesis of one-shot realistic head avatars. In *European Conference of Computer vision (ECCV)*, August 2020.

[163] 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.

[164] 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.

[165] Zhengdong Zhang and Vivienne Sze. FAST: A framework to accelerate super-resolution processing on compressed videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 19–28, 2017.

[166] Zhimeng Zhang, Lincheng Li, Yu Ding, and Changjie Fan. Flow-guided one-shot talking face generation with a high-resolution audio-visual dataset. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3661–3670, June 2021.

[167] Daquan Zhou, Bingyi Kang, Xiaojie Jin, Linjie Yang, Xiaochen Lian, Zihang Jiang, Qibin Hou, and Jiashi Feng. Deepvit: Towards deeper vision transformer. *arXiv preprint arXiv:2103.11886*, 2021.

[168] Hang Zhou, Yasheng Sun, Wayne Wu, Chen Change Loy, Xiaogang Wang, and Ziwei Liu. Pose-controllable talking face generation by implicitly modularized audio-visual representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4176–4186, June 2021.

[169] Shangchen Zhou, Kelvin CK Chan, Chongyi Li, and Chen Change Loy. Towards robust blind face restoration with codebook lookup transformer. *arXiv preprint arXiv:2206.11253*, 2022.

[170] Tinghui Zhou, Shubham Tulsiani, Weilun Sun, Jitendra Malik, and Alexei A Efros. View synthesis by appearance flow. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV 14*, pages 286–301. Springer, 2016.

[171] Yang Zhou, Xintong Han, Eli Shechtman, Jose Echevarria, Evangelos Kalogerakis, and Dingzeyu Li. Makelttalk: Speaker-aware talking-head animation. 39(6), nov 2020.

Appendix A

Gemino Model Details

In the following subsections, we detail the structure of the motion estimator that produces the warping field for Gemino and the neural encoder-decoder pair that produce the prediction. We also describe additional details about the training procedure.

A.1 Motion Estimator

UNet Structure. The keypoint detector and the motion estimator use identical UNet structures (Fig. A-1 and Fig. A-2) to extract features from their respective inputs before they are post-processed. In both cases, the UNet consists of five up and downsampling blocks each. Each downsampling block consists of a 2D convolutional layer [56], a batch normalization layer [72], a Rectified Linear Unit Non-linearity (ReLU) layer [88], and a pooling layer that downsamples by 2× in each dimension. The batch normalization helps normalize inputs and outputs across layers, while the ReLU layer helps speed up training. Each upsampling block first performs a 2× interpolation, followed by a convolutional layer, a batch normalization layer, and a ReLU layer. Thus, every downsampling layer reduces the spatial dimensions of the input but instead extracts features in a third "channel" or "depth" dimension by doubling the third dimension. On the other hand, every upsampling layer doubles in each spatial dimension, while halving the number of features in the depth dimension. In our implementation, the UNet structure always produces 64 features after its first encoder downsampling layer, and doubles from there on. The reverse happens with the decoder ending with 64 features after its last layer. Since the UNet structure operates on low-resolution input (as part of the keypoint detector and motion estimator), its kernel size is set to 3×3 to capture reasonably sized fields of interest.

Keypoint Detector. To obtain the warping field between the reference frame and the target, Gemino first uses a keypoint detector to locate key facial landmarks. It then uses a first-order approximation in the neighborhood of these keypoints similar to the FOMM [132]. To extract keypoints, we first downsample the input image to 64×64,

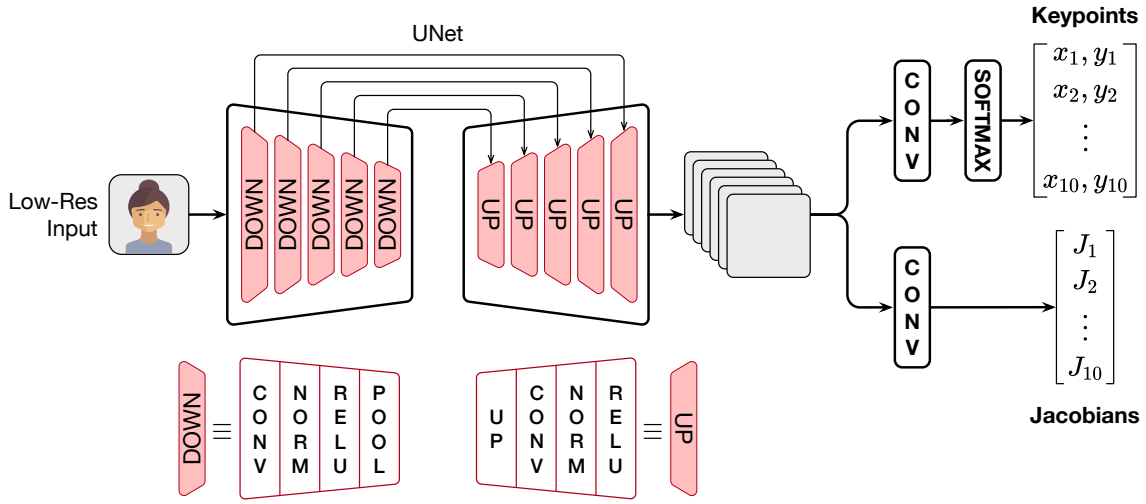


Figure A-1: Keypoint Detector used as a precursor for computing the warping field between the reference and target images. Low-resolution versions of both frames are supplied to a UNet architecture [124], and then put through convolutional layers to generate keypoint locations and four "Jacobian" values in the neighborhood of each keypoint.

and then feed it into the UNet structure described above in its RGB space itself. The UNet structure produces a set of output features from its decoder, which are then put through two separate pipelines to extract the keypoint locations and the "jacobians." The keypoint locations are extracted via a single 7×7 convolutional layer, which is then put through a softmax to extract probabilities for keypoint presence at each spatial location. This is then converted to actual keypoint locations by performing a weighted average of these probabilities across the entire spatial grid. Note that this process is replicated 10 times by having 10 separate channels to extract 10 keypoints. The jacobians are simply four floating point numbers that are used to approximate the movement (derivatives) in the neighborhood of each keypoint. This is used for the first-order approximation when computing the motion around each keypoint. To generate these jacobians, the output from the UNet is simply put through a single 7x7 convolutional layer. Fig. A-1 describes this architecture. Note that both the reference and the target images are fed to this pipeline independently to generate two separate sets of reference and target keypoints and jacobians.

Motion Estimation Fig. A-2 describes the working of the motion estimator in Gemino's design in detail. First, the motion estimator creates Gaussian *heatmaps* corresponding to the keypoint locations from both the reference and the target frames. It subtracts the two on a per-keypoint basis to generate the difference between the two frames' keypoints. It adds a separate heatmap consisting of zeros to denote the fact that the background is identical in the two frames. The motion estimator then generates *sparse motion* vectors or motion vectors

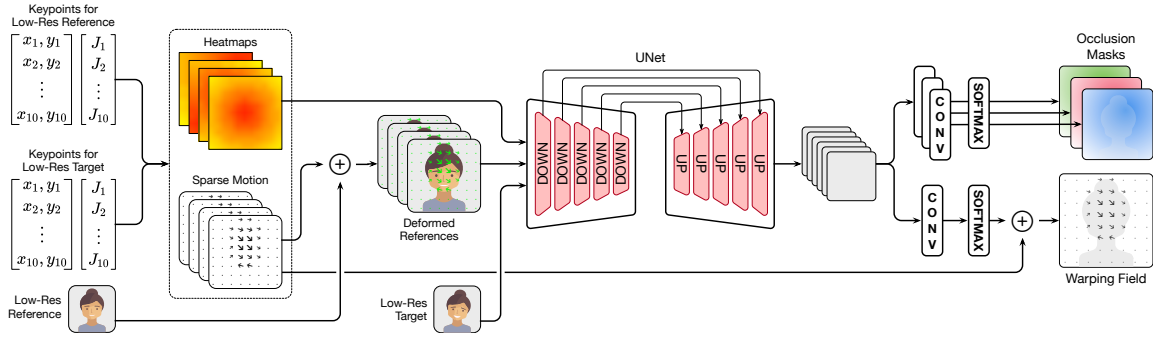


Figure A-2: Gemino's motion estimation module that takes as input reference and target keypoints, along with a low-resolution reference and target frames, and produces a warping field and occlusion masks. The warping field helps the generator move encoded features from a full-resolution reference frame into the target frame's coordinate space, while the occlusion masks inform the decoder how to combine information from the low-resolution target frame with high-resolution (warped and unwarped) features from the reference frame.

in the neighborhood of each keypoint using the first-order Taylor series approximation [132] and the Jacobian values from the keypoint detector. These motion vectors (along with an identity for the background) are applied to the low-resolution reference frame to obtain a set of *deformed references*. This effectively generates 11 heatmaps (10 keypoints + 1 for background), and 11 different RGB (3 channels) deformed references. The 44 resulting channels are provided as input along with 3 RGB features from the low-resolution target image to another replica of the UNet structure described above. This UNet's decoder also outputs a set of predicted features based on all the provided 47 input features.

The predicted features are put through three separate 7×7 convolutional layer followed by Sigmoid layers and a Softmax layer to produce *three occlusion masks*. Each occlusion mask is later used in the decoding pipeline to convey how to combine information from three pathways: the warped high-resolution features, the non-warped high-resolution features, and the low-resolution features to generate the prediction. We use a Softmax layer to enforce that the sum of these three occlusion masks is 1 at every spatial location so that they do not compete in later parts of the decoding pipeline. Intuitively, this forces each pixel to be generated from one out of the three pathways. If a feature represents a part of the frame that has moved between the reference and the target frames, reconstruction relies on the HR warped pathway, while if it represents a part of the frame that has not moved, it relies on the non-warped HR pathway. Regions that are significantly different between the reference and the target use the LR features instead.

The predicted features are also fed into a single 7×7 convolutional layer followed by a Softmax to generate a *deformation mask* that reflects how to combine the sparse motion

vectors previously obtained in the neighborhood of each location. The Softmax ensures that across every spatial location, different vectors are weighted appropriately to sum up to 1 finally. This deformation mask is applied to the sparse motion vectors to obtain the final warping field.

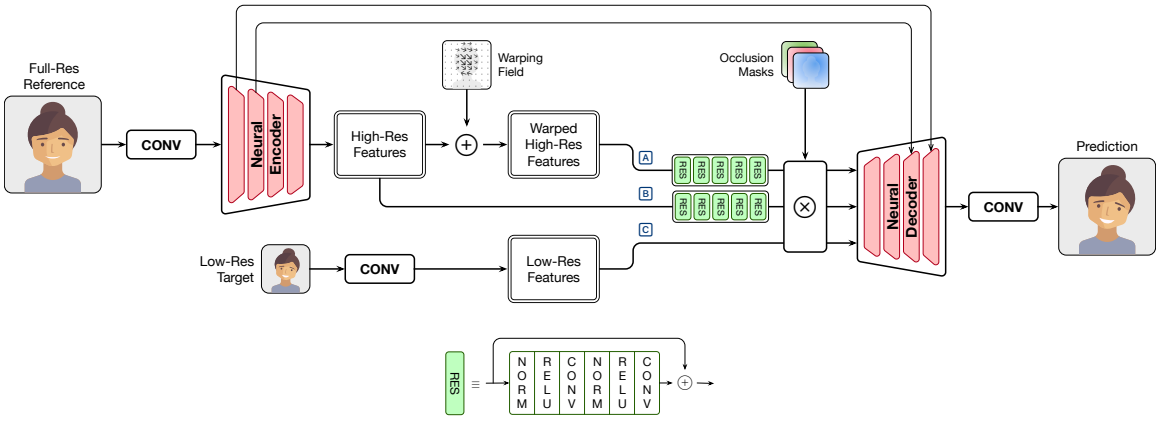


Figure A-3: Gemino's encoder-decoder pair that is responsible for synthesizing the prediction. The encoder runs the high-resolution reference image through a series of downsampling layers to produce encoded features. A copy of these encoded HR features is warped, and both the warped and non-warped features are refined through a sequence of residual blocks. Meanwhile, a convolutional layer extracts low-resolution features from a low-resolution target. The three sets of features are combined based on occlusion masks from the motion estimator before they are decoded to result in the final prediction.

A.2 Image Synthesis

Fig. A-3 describes the generative parts of Gemino design in more detail. First, the low-resolution target frames are put through a single 7×7 convolutional layer to produce low-resolution (LR) features to be used later during decoding. The high-resolution reference RGB frame is also fed through a single 7×7 convolutional layer to produce 32 high-resolution (HR) features with the same spatial dimensions before it is fed to the neural encoder. The neural encoder consists of four downsample blocks, each of which has the same structure as the downsample blocks in our UNet structure in the keypoint detector (Fig. A-1). The four blocks ensure that we start at full-resolution (1024×1024 frames) with 32 features and end up with 256 separate 64×64 encoded HR features at the bottleneck. However, unlike the UNet, not all blocks in the neural encoder are equipped with skip connections. Specifically, only the first two have skip connections to the corresponding last two decoder blocks. The decoder also consists of upsampling blocks with the same composition as the UNet's downsampling blocks. Prior to decoding, a copy of the HR features are warped using the warping field from the motion estimator (Ⓐ). The warped HR features and the unwarped

YouTuber	Training Videos		Test Videos	
	Total Len.	Avg. Bitrate	Avg. Len.	Avg. Bitrate
Adam Neely	31 min	1082 kbps	146 s	1303 kbps
Xiran Jay Zhao	30 min	2815 kbps	180 s	1560 kbps
The Needle Drop	33 min	2013 kbps	206 s	1286 kbps
fancy fueko	15 min	4064 kbps	124 s	1607 kbps
Kayleigh McEnany	28 min	2521 kbps	180 s	1247 kbps

Table A-1: Details of our dataset. All videos are at 1024×1024.

features (Ⓑ) are fed through five Residual Blocks [65] that refine them and also help prevent diminishing gradients during training. Each residual block consists of batch normalization, ReLU, and convolutional layers. The refined HR features, along with the LR features, and appropriate skip connections are combined using occlusion masks obtained from the motion estimator. Once combined, all three sets of features are fed through decoder's upsampling blocks to spatial dimensions of 1024×1024 frames with 32 features. This output is fed through a final 7×7 convolutional layer to bring it back to its RGB space as the prediction.

A.3 Training Details

To train our model, we use equally weighted (with a weight 10) multi-scale VGG perceptual loss [75], a feature-matching loss [148], and an L1 pixel-wise loss. The multi-scale VGG loss and feature matching losses operate at scales of 1, 0.5, 0.25, and 0.125 of the image size. All of these scales have equal loss weights. We also use an adversarial loss [59] with a weight of one (one-tenth of the other losses). The keypoints use an equivariance loss similar to the FOMM [132]. Our keypoint detector is unchanged relative to the FOMM, and so we reuse a trained checkpoint on the VoxCeleb dataset [111], but fine-tune it on a per-person basis. For other layers, we adopt a simple strategy: if the dimensions match the equivalent layers in the FOMM (*e.g.,* encoder layers, residual layers), we initialize them based on a FOMM checkpoint, and if the dimensions do not match, they are initialized randomly. All models are personalized (either trained from scratch or fine-tuned if the layers are initialized to FOMM checkpoints). All models for 1024×1024 output resolution were trained with a batch size of 2 on NVIDIA A100 GPUs while models for 512×512 output resolution were trained with a batch size of 2 on NVIDIA V100 GPUs. The details of the dataset we curate is shown in Tab. A-1 As discussed earlier, models are trained on decompressed VPX frames at different bitrates. To support this, we encode individual frames at the target bitrate during train-time *after* they are sampled from the dataset. Since we encode individual frames, this only results in keyframes at train-time, but these frames are still representative of what compressed video frames more generally look like at that bitrate.

Appendix B

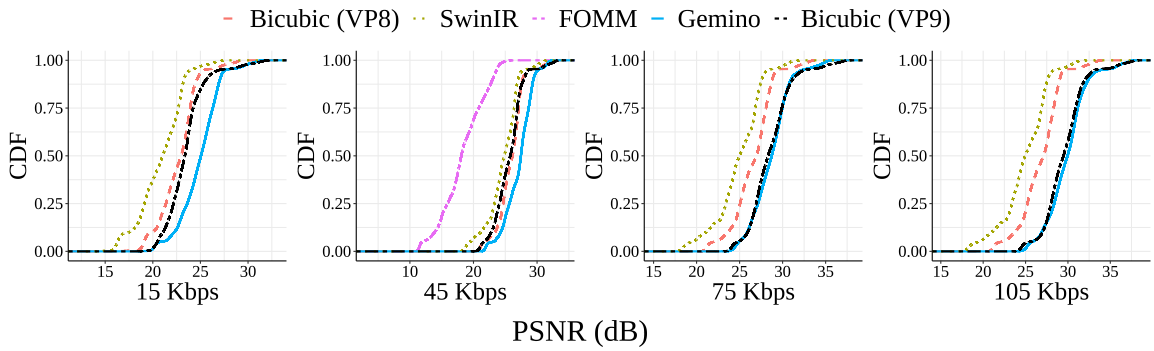
Extended Gemino Evaluation

B.1 Low-bitrate Regime Comparisons

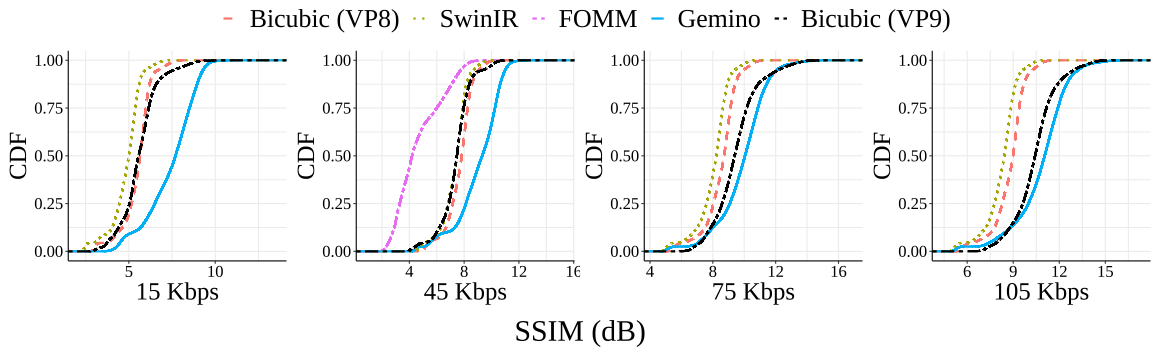
We plot a CDF of the visual quality in the context of PSNR and SSIM across all frames in our corpus in Fig. B-1 at 15 Kbps, 45 Kbps, 75 Kbps and 105 Kbps. At each target bitrate, we use the largest resolution supported by the underlying video codec. Gemino uses VP8 at 15 Kbps and 45 Kbps but VP9 at higher bitrates; VP8 and VP9 do not differ much at lower bitrates, but VP9 allows 512×512 to be compressed all the way to 75 Kbps while VP8 cannot. Since our results in §3.4.4 suggest using the highest resolution at any target bitrate, we use VP9 at 75 Kbps and 105 Kbps. The CDF shows that Gemino's reconstructions are robust to variations across frames and orientation changes over the course of a video. At 45 Kbps when upsampling from a 256×256 frame, Gemino outperforms all other baselines across all frames and both metrics. Specifically, its synthesized frames are better than FOMM by nearly 5 dB in SSIM and 10 dB in PSNR throughout. While the differences in PSNR between Bicubic atop VP9 and Gemino are small at 105 Kbps, the difference at the median is over 2 dB at 15 Kbps. The differences in SSIM is even more pronounced at 15 Kbps with Gemino outperforming both bicubic approaches by nearly 2.5 dB at the median. As the bitrate is lowered, the differences between VP8 and VP9 also shrink, resulting in very similar visual quality for both of their bicubic upsampling approaches. As we show in §B.2, differences in PSNR are not as reflective of how natural the image looks to the human eye, and so we rely on LPIPS [163] primarily, and SSIM when relevant.

B.2 Comparing the metrics.

To understand the difference between different quantitative metrics and the visual quality each of them map to, we show a strip (with the associated metrics) that compares Gemino with the reconstruction of bicubic atop VP8 and VP9 when upsampling 256×256 frames at 45 Kbps in Fig. B-2, B-3, B-4 and B-5. For example, in Fig. B-2, while the



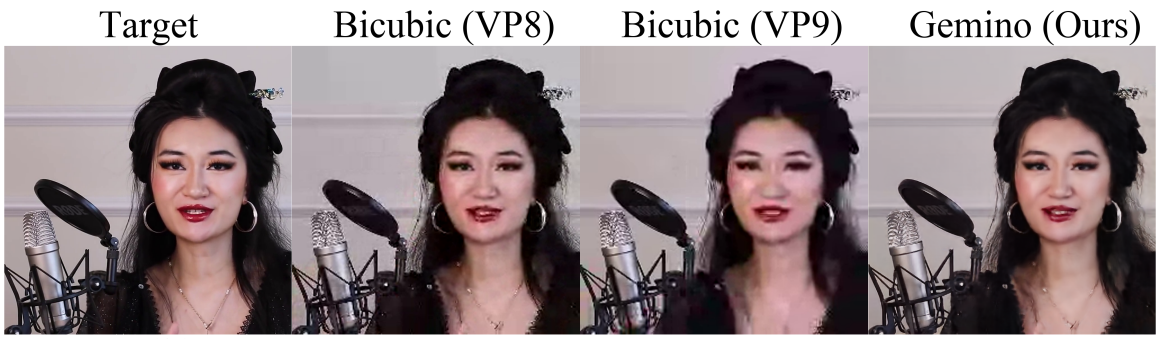
(a) PSNR achieved by different schemes in the low-bitrate regime



(b) SSIM achieved by different schemes in the low-bitrate regime

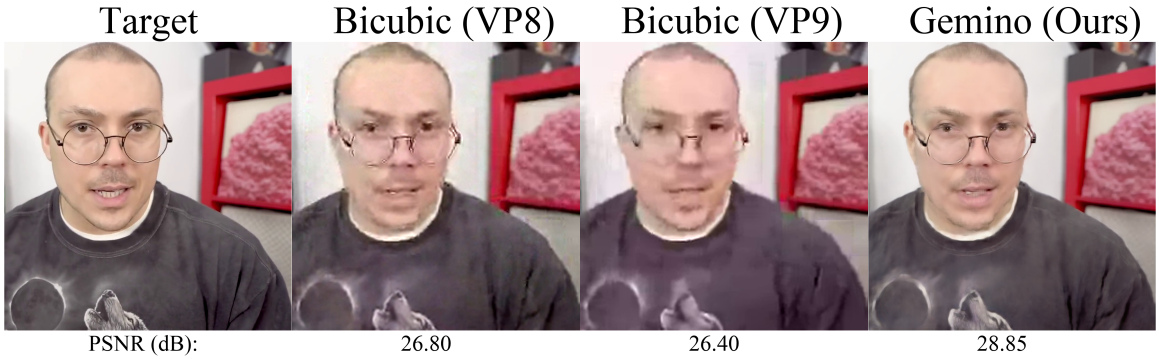
Figure B-1: CDF of PSNR and SSIM achieved by different schemes on all frames in low-bitrate regimes. The differences between Gemino and other approaches becomes more pronounced in lower-bitrate regimes.

output produced by Gemino is significantly better than that of both bicubic approaches, it manifests as only a 2 dB difference in PSNR and SSIM, both of which have very wide ranges. In contrast, we see an improvement of 0.1 in LPIPS. LPIPS is constrained to be between 0 and 1, making it easier to put the 0.1 improvement in context. We see similar trends in Fig. B-3 and in Fig. B-4, where a difference of 0.13-0.14 in LPIPS, and $\sim$2 dB PSNR manifests as smoother output on the facial regions. In contrast, while the PSNRs in Fig. B-5 are fairly close across the bicubic approaches and Gemino, the SSIM and LPIPS differences reflect how much more natural Gemino's facial reconstruction looks. Frames like these explain why the PSNR graphs in Fig. 3-5 are a lot closer than the other two metrics, and motivate the use of LPIPS as our main metric. However, even though the two bicubic approaches in Fig. B-3 differ significantly in the smoothness of the face, all three metrics are close enough that the difference would not be very perceptible on a plot. As a result, we acknowledge that none of these metrics are truly perfect, and a combination of them all along with visual strips is needed for a thorough evaluation.



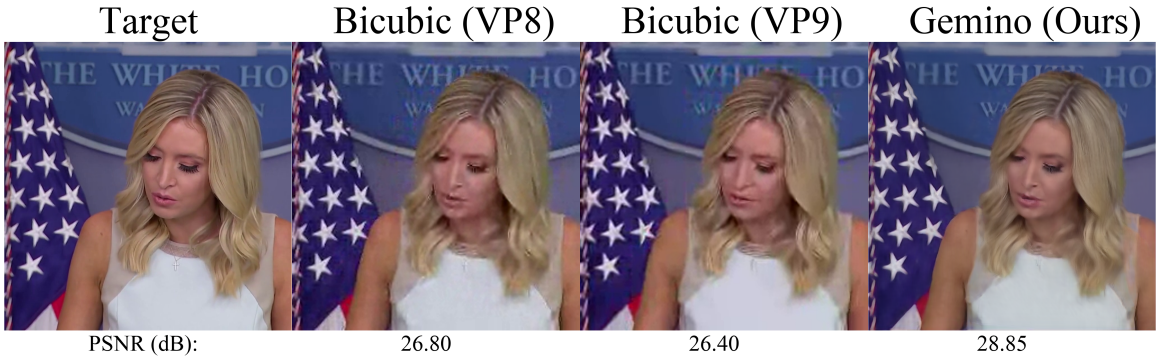
PSNR (dB):	25.40	24.71	27.82
SSIM (dB):	7.81	7.42	10.10
LPIPS:	0.29	0.30	0.19

Figure B-2: Full-size examples to illustrate the visual quality differences and their correlation with the metric values.



PSNR (dB):	26.80	26.40	28.85
SSIM (dB):	7.44	7.22	9.91
LPIPS:	0.34	0.33	0.20

Figure B-3: Full-size examples to illustrate the visual quality differences and their correlation with the metric values.



PSNR (dB):	26.80	26.40	28.85
SSIM (dB):	7.44	7.22	9.91
LPIPS:	0.34	0.33	0.20

Figure B-4: Full-size examples to illustrate the visual quality differences and their correlation with the metric values.

PSNR (dB):	25.47	24.59	25.64
SSIM (dB):	8.31	7.46	9.29
LPIPS:	0.28	0.31	0.24

Figure B-5: Full-size examples to illustrate the visual quality differences and their correlation with the metric values.