Gemino: Practical and Robust Neural Compression for Video Conferencing

Vibhaalakshmi Sivaraman, Pantea Karimi, Vedantha Venkatapathy, and Mehrdad Khani, *Massachusetts Institute of Technology;* Sadjad Fouladi, *Microsoft Research;* Mohammad Alizadeh, Frédo Durand, and Vivienne Sze, *Massachusetts Institute of Technology*

https://www.usenix.org/conference/nsdi24/presentation/sivaraman

This paper is included in the
Proceedings of the 21st USENIX Symposium on
Networked Systems Design and Implementation.

April 16–18, 2024 • Santa Clara, CA, USA

978-1-939133-39-7

Gemino: Practical and Robust Neural Compression for Video Conferencing

Vibhaalakshmi Sivaraman Pantea Karimi Vedantha Venkatapathy Mehrdad Khani
Sadjad Fouladi ⊞ Mohammad Alizadeh Frédo Durand Vivienne Sze

Massachusetts Institute of Technology, ⊞ *Microsoft Research*

Abstract

Video conferencing systems suffer from poor user experience when network conditions deteriorate because current video codecs simply cannot operate at extremely low bitrates. Recently, several neural alternatives have been proposed that reconstruct talking head videos at very low bitrates 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. We design 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. We use a multi-scale architecture that runs different components of the model at different resolutions, allowing it to scale to resolutions comparable to 720p, and we 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.

1 Introduction

Video conferencing applications have become a crucial part of modern life. However, today's systems continue to suffer from poor user experience: in particular, poor video quality and unwelcome disruptions are all too common. Many of these problems are rooted in the inability of today's applications to operate in low-bandwidth scenarios. For instance, Zoom recommends a minimum bandwidth of 1.2 Mbps for one-on-one meetings and 2-3 Mbps for group meetings [1]. In certain parts of the world, Internet broadband speeds remain insufficient for reliable video conferencing. For example, large swaths of the population in Africa and Asia had average Internet broadband speeds less than 10 Mbps in 2022 [2], with the five slowest countries having speeds under 1 Mbps. Mobile bandwidth is even more restricted: SpeedTest's Global Index [3] suggest that global mobile bandwidth average is 50% of broadband speeds. Even in regions of North America and Europe with high broadband speeds [2], over 30% of users surveyed about their video conferencing experience claimed that "video quality" issues were their biggest pain point [4]. This is because the user experience is not just determined by the average bandwidth, but rather by tail events of low bandwidth (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.

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) [5–9]. These techniques have the potential to enable video conferencing with one to two orders of magnitude reduction in bandwidth (as low as ∼10 Kbps [6, 8]), 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 [5, 8], even on high-end GPUs, while typical video conference applications are designed for HD, Full HD, and even 4K videos [10, 11]. Naïvely reusing these models on larger input frames can quickly become prohibitively expensive as the resolution is increased.

We present Gemino, a neural compression system for low-bitrate video conferencing, designed to overcome the above robustness and compute complexity challenges. 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 [12], greatly expanding the range of networks that can support video conferencing.

Gemino's design begins with the observation that current

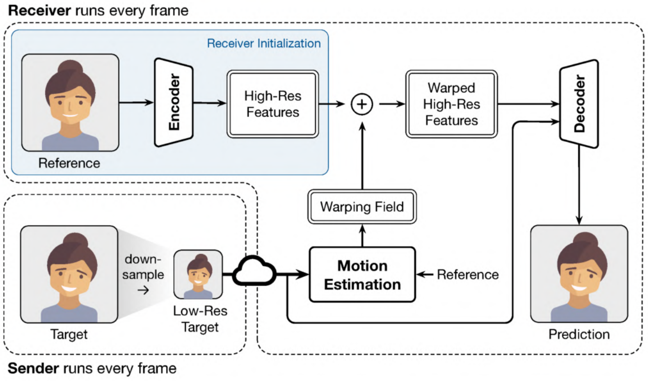


Figure 1: Gemino's design. The sender sends a downsampled version of the target frame across the network to the receiver that has a reference image of the speaker in the current video conference setting. The encoder network encodes and warps features from the reference image based on the motion estimated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

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 (§2). 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 [13–16] compress them very efficiently. For example, sending 128×128 resolution video in our implementation consumes ~15 Kbps, only slightly more than would be needed to transmit keypoints [6, 8]. 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). We call this new approach *high-frequency-conditional super-resolution*.

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 [18] to reduce the compute footprint.

We implement and evaluate Gemino within *aiortc* [19], a Python implementation of WebRTC, and show the following:

1. Gemino achieves a perceptual quality (LPIPS) [20] 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 [21] 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 (~0.05 in LPIPS). Our optimizations atop the multi-scale architecture enable real-time inference on 1024×1024 frames on a Titan X GPU.

[1] "Encoder" and "decoder" throughout this paper refer to the neural network pair typically used in GANs [17]. 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.

2 Related Work

Traditional Codecs. Most video applications rely on standard video compression modules (codecs) such as H.264/H.265 [22,23], VP8/VP9 [24,25], and AV1 [13]. 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. Over the years, these standards have been improved through ideas like variable block sizes [23] and low-resolution encoding for lower bitrates [13]. These codecs are particularly efficient in their *slow* modes when they have generous time and compute budget to compress a video at high quality. 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 [26]. To circumvent this, some applications [27] switch to lower resolutions when the network degrades. However, as new video conferencing solutions such as Google's Starline [11] with a large bandwidth footprint are introduced, these concerns with current codecs become more acute.

Super-resolution. Linear single-image super-resolution (SR) methods [28, 29] provide robust quality enhancements in various contexts. Neural SR methods have further enhanced the upsampling quality by learning better interpolation or in-painting methods [21,30–32]. Video SR methods [33,34] build on image SR but further improve the reconstruction by exploiting redundant information in adjacent low-res video frames. Certain approaches like FAST [35] and Nemo [36] 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 [37,38]. 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 provides access to high-resolution reference frames and learns models that jointly in-paint and propagate high-frequency details from the reference frame. In recent work, SRVC [39] 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.

Neural Codecs. The inability of traditional codecs to operate at extremely low bitrates for high-resolution videos has led researchers to consider neural approaches that reconstruct video from very compact representations. Neural codecs have been designed for video-streaming [39–41], live-video [42], and video video conferencing [6,8]. Swift [41] learns to compress and decompress based on the residuals in a layered-encoding stack. Both NAS [40] and LiveNAS [42] 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) [40], or by adjusting the number of epochs over which they are fine-tuned online [42]. All of these approaches have shown improvements in the bits-per-pixel 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 [5–9,43,44] 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 [6], off-the-shelf keypoint detectors [7], or multiple reference frames [9] to enhance prediction.

Challenges for neural face image synthesis. 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 but do not transmit or use the downsampled target frame. They extract keypoints from the downsampled target frame, and transmit those instead. This choice causes major reconstruction failures when the reference and target frames are not close. Fig. 2 shows the reconstruction produced by the First-Order-Motion Model (FOMM) [5], 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. 2, 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. 2, 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 frame,

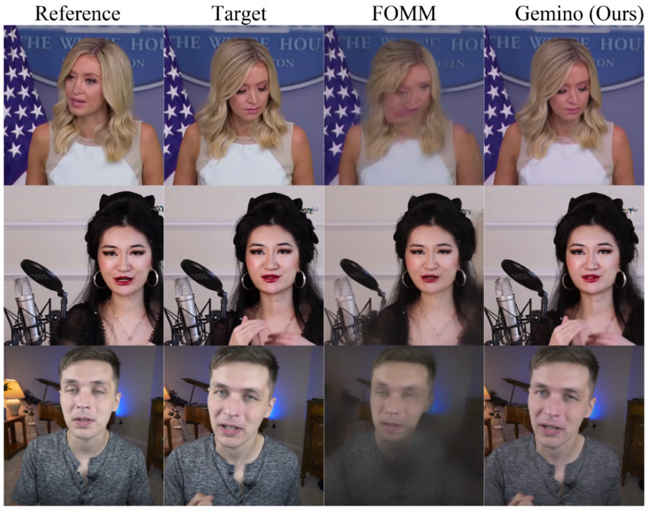


Figure 2: Failure cases in the FOMM's [5] reconstruction when the reference and target differ. FOMM misses the hand (row 2) because the reference frame does not contain it. In rows 1 and 3, FOMM only produces a blurry outline of the face (and torso). Gemino improves on these reconstructions by utilizing the downsampled target frame in its architecture and capturing the difference between the reference and the target better.

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

Lastly, many existing approaches [5, 8, 9] are evaluated on 256×256 images [45]. 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. 1 which shows the inference time of the FOMM [5] 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 System Design

3.1 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.

	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 1: Inference time of the First-Order-Motion Model [5] 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.

3.2 Model Architecture

Fig. 3 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 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 [46]. 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: $\boxed{A}$ the warped HR features, $\boxed{B}$ the HR features prior to warping, and $\boxed{C}$ 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

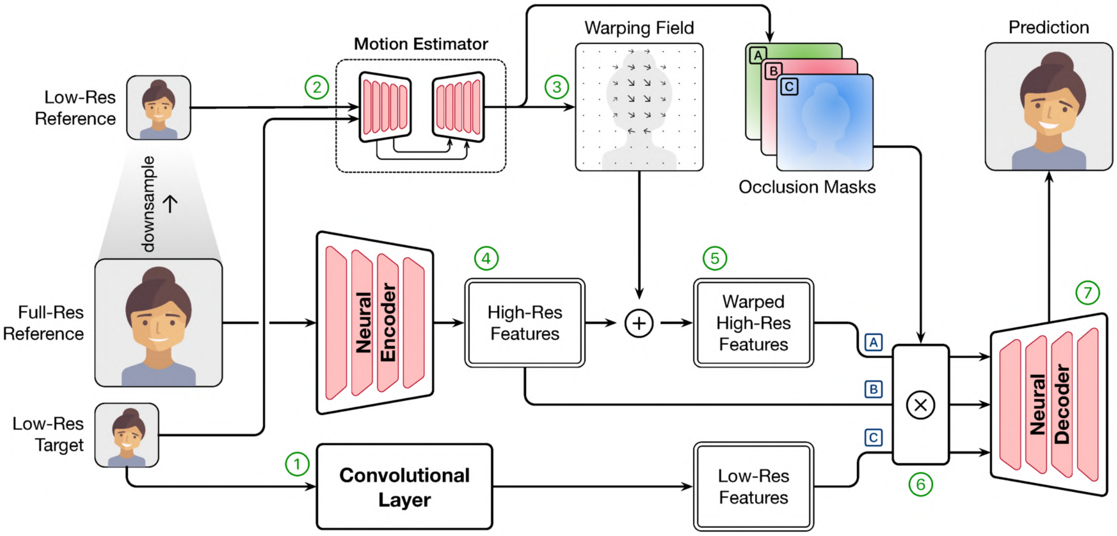


Figure 3: Gemino's high-frequency-conditional super-resolution model at the receiving client. The model first obtains features from the low-resolution target. It combines the low-resolution reference and target frames to produce a warping field based on the motion between them. The warping field is applied 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.

different regions of the input features when reconstructing each region of target frame. For instance, mask A maps to parts of the face or body that have moved in the reference and need to be regenerated in the target pose, mask B maps to the regions that don't move between the reference and target (*e.g.,* background), while mask C 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 (§5.3). The former ensures that we good low-frequency fidelity, while the latter handles the high-frequency details. The last column of Fig. 2 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.

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

3.3 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 [24, 25, 47] 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. 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)

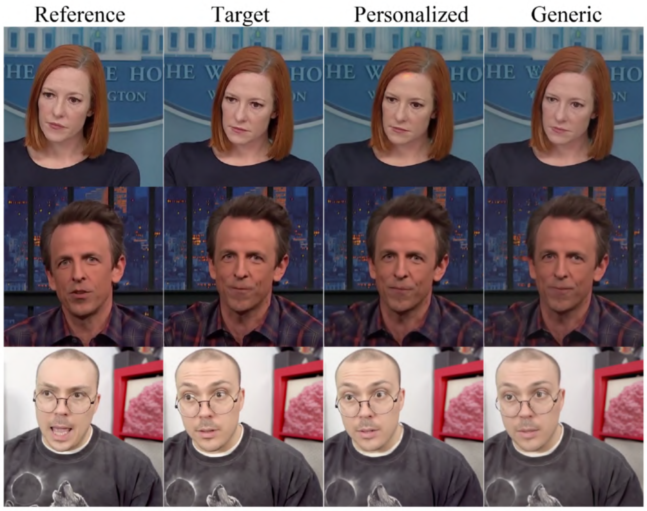


Figure 4: Performance of a model trained on a generic corpus of 512×512 videos compared to a personalized model fine-tuned on the specific person in the video. The personalized model better captures the eye gaze, dimples, and rim of the glasses.

produced by the codec at that particular 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 (§5.4).

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. 4 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 §5.4.

3.4 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, 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 adjust the number of downsampling and upsampling blocks in the neural encoder and decoder based on the input resolution, keeping the size of the intermediate bottleneck tensor manageable. For example, for 1024×1024 input, we use 4 down/up-sampling blocks in the encoder and decoder (Fig. 3). So as to not lose information through the bottleneck, we equip the first two blocks with skip connections [46] that directly provide encoded features to the decoder. As seen in the third column of Tab. 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 [18], 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 [48, 49], a commonly used architecture for mobile inference.

To decrease the model MACs even further without impacting its fidelity, we run NetAdapt [18], 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 prunes the layer of the model which decreases its MACs with the least loss in accuracy after "short-term finetuning" on a small subset of the data. The pruned model acts as the starting point for the next iteration, and this process continues until the model has been shrunk sufficiently. Finally, the model is "long-term fine-tuned" on the entire dataset to recover its lost accuracy.

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 (§5.4). These optimizations allow us to run inference on the Titan X GPU in 27 ms with barely any loss in visual quality (last column of Tab. 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 [50], we believe that the additional delay from generating the received frame will be negligible.

3.5 Operational Flow

Training Procedure. To train Gemino, we first obtain weights from a trained FOMM model on the entire VoxCeleb dataset [45] 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 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. 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 [18] 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 §4.

4 Implementation

Basic WebRTC Pipeline. Our neural video conferencing solution uses WebRTC [51], an open-source framework that enables video and audio conferencing atop the real-time transport protocol (RTP) [52]. Since we perform neural frame synthesis, we use a Python implementation of WebRTC called *aiortc* [19] 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 [24, 25], H.264/5 [22, 23], Opus [53], *etc.*. The receiver decompresses the received data in both streams before synchronizing them and displaying each frame to the client.

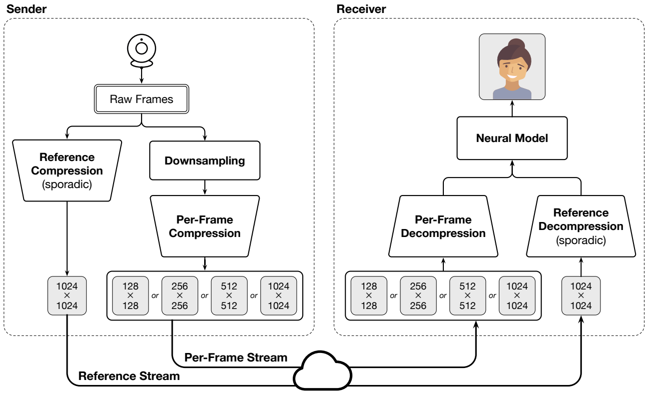


Figure 5: Neural video compression pipeline atop WebRTC [51]. We use two RTP streams: A sparse reference stream that sporadically sends high-resolution reference frames, and a per-frame (PF) stream that is used on every frame. The PF stream sends downsampled frames of the highest resolution that the 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.

New Streams. We extend the standard WebRTC stack to use two distinct streams for 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. But, most low-frequency changes between the reference and the target can be communicated simply through the downsampled target in the PF stream[3]. The receiver uses the per-frame information in the PF stream, with the reference information, to synthesize each high-resolution frame. Fig. 5 illustrates the expanded WebRTC architecture to accommodate 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

[3] 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.

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 [54] frames on the CPU that *aiortc* needs, to the PyTorch [55] 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.

5 Evaluation

We evaluate Gemino in a simulation environment and atop a WebRTC-based implementation. We describe our setup in §5.1 and use it to compare existing baselines in §5.2. §5.3 motivates our model design, §5.4 discusses the impact of having the codec in our training process, and §5.5 shows that Gemino closely matches a time-varying target bitrate.

5.1 Setup

Dataset. Since most widely used datasets are of low-resolution videos [6, 45, 56] 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 720p and 1024×1024 frames have similar numbers of pixels. We strip the audio since our focus is on video synthesis. Tab. 8 in App. A.3 describes the details of the dataset. We do not own any of these videos, and we only use images of frames produced by our evaluation pipeline in this paper. We use the 512×512 dataset from NVIDIA [6] 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 [57, 58].

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). 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 [5]. 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 [59] 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 [45], 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 [60] 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 [61], a feature-matching loss [62], and a pixel-wise loss. We also use an adversarial loss [17] with one-tenth the weight of remaining losses. The keypoints use an equivariance loss similar to the FOMM [5]. 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.

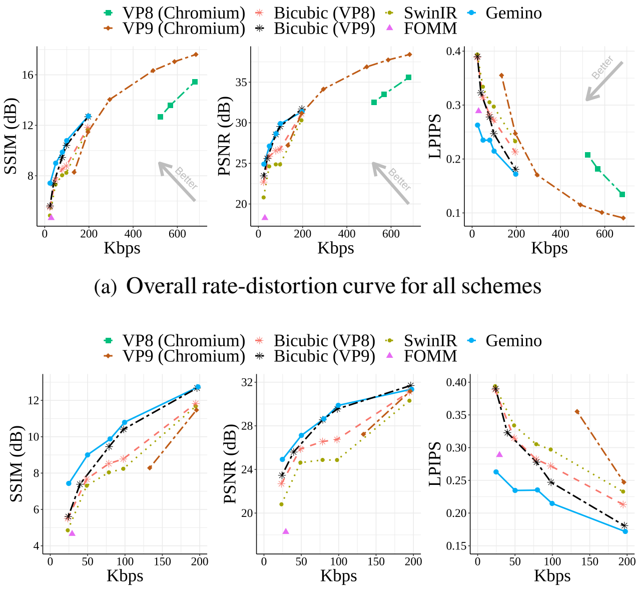
Evaluation Infrastructure. We evaluate our neural compression system in a simulation environment where frames are read from a video, downsampled (if needed) for the low-resolution

PF stream, compressed using VPX's chromium codec [63], and passed to the model (or other baselines) to synthesize the target frame. Note that the FOMM [5] 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, (§5.5), we use our *aiortc* [19] 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 [64] 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 aesthetics of the generated video, we use standard visual metrics such as PSNR (peak signal-to-noise ratio), SSIM (structural similarity index) in decibels [65], and LPIPS (learned perceptual image patch similarity) [20]. 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. 8), and report the average over all frames for each metric.

Baselines. We obtain the bitrate for *VP8*, the default codec in its Chromium settings [63] 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* [5], a keypoint-based model for face animation, (2) our approach, Gemino, (3) state-of-the-art super-resolution model based on *SwinIR* [21], and (4) *bicubic upsampling* [28] applied to the low-resolution VPX target frame. All of the compared models generate 1024× 1024 frames except for the generic model that uses NVIDIA's 512×512 corpus [6].



(a) Overall rate-distortion curve for all schemes

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

Figure 6: 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.

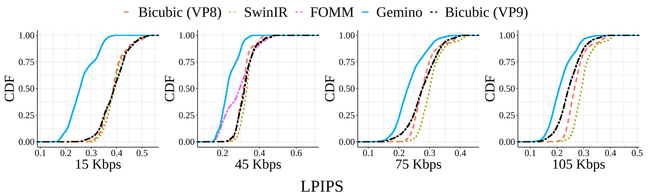


Figure 7: CDF of reconstruction quality across all video frames as that shows that, as we move from higher bitrates to lower, the improvement from Gemino relative to Bicubic, particularly over VP9, becomes more pronounced.

5.2 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 [63]. Fig. 6 shows the rate-distortion curve for all schemes. For VPX, we alter the target bitrate alone for full-resolution (1024×1024) 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

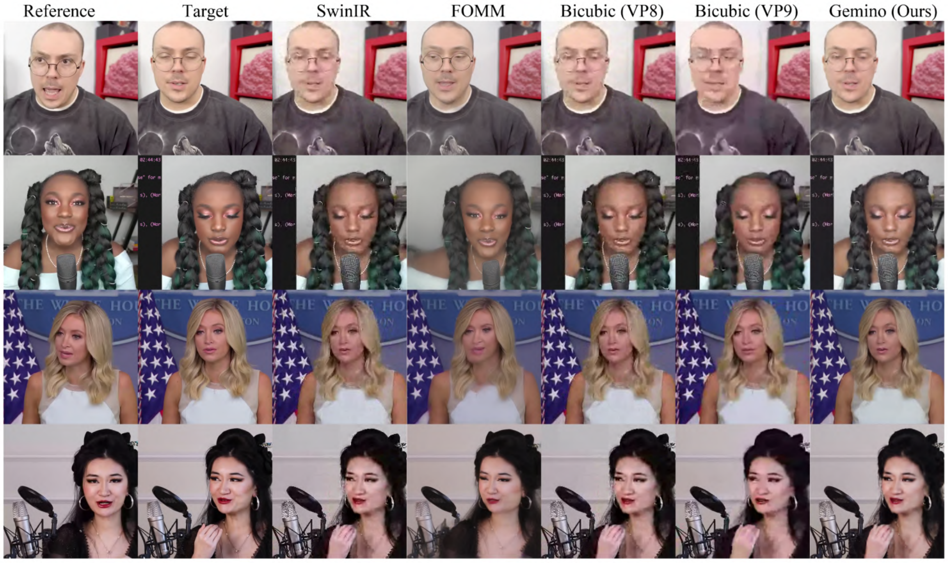


Figure 8: Visual comparison across low-bitrate baselines. All but FOMM upsample a 256×256 frame at 45 Kbps. Gemino's reconstructions contain a smoother output than the blocky artifacts observed with Bicubic and SwinIR. The FOMM completely fails when the reference and the target differ considerably in rows 1 and 2.

the codec in training and choosing the resolution in §5.4. 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. 6(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 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. 6(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. 6(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 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. 8 and enlarged frames (with per-frame metrics) in §B.2. Fig. 8 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 §5.3.

We also show CDFs of the visual quality in LPIPS across all frames in our corpus in Fig. 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 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.[4]

5.3 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) at decompressing VP8 128×128 frames at 15 Kbps to its full 1024×1024 resolution. Tab. 3 reports the average visual quality while Fig. 10 shows a CDF of performance across all frames and videos. Gemino

[4] 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 [54] is not optimized for conversion at higher resolutions.

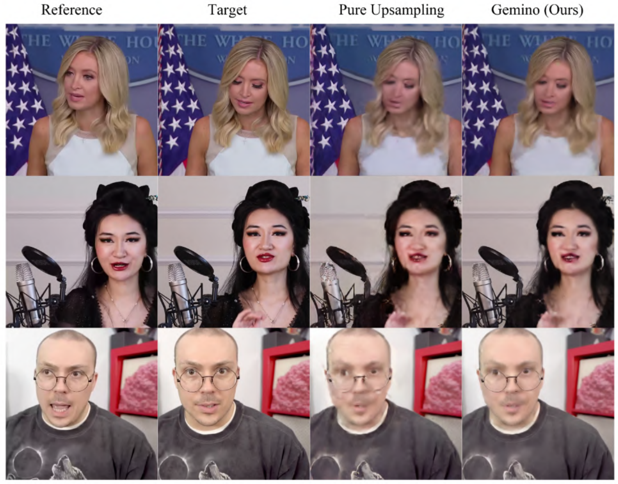


Figure 9: Visual comparison across different model architectures. Compared to Gemino, Pure upsampling misses the high-frequency details of the microphone (row 2) and lends block-based artifacts on the face (row 3).

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

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

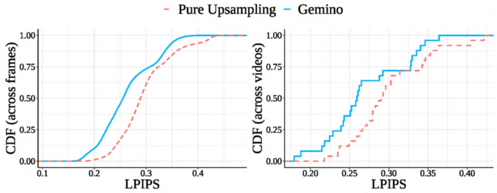


Figure 10: CDF of reconstruction quality of different model architectures across frames and videos in our test corpus. Gemino outperforms pure upsampling by nearly 0.05 in LPIPS at the median of both frames and videos. It also outperforms the approach that relies on only warped HR, and the RGB-based warping across most frames in the corpus.

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. 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. 10. Since one 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 particularly salient. On the other hand, Gemino synthesizes the mic 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 [6] 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. 4 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. 4 shows that personalization provides limited benefits 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. 10).

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 4: Impact of personalization on different models.

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

5.4 Operational Considerations

Computational Overheads. Tab. 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 §3.4. We focus on optimizing the decoding layers (bottleneck and up-sampling) of the generator (Fig. 14) 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 [20] across all frames of our corpus. We observe that DSC reduces the decoder to 11% of its original MACs. While this gives limited improvements on large GPU systems, it improves the inference time on Jetson TX2, an embedded AI device, by 1.84×. 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 GPU and the Jetson TX2 is not optimized for DSC [49]; this can be improved with a TVM compiler stack [66] and optimized engines such as TensorRT [67]. 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%, particularly when personalizing, but is more significant at 1.5%. The trend with personalization is expected since smaller models do not generalize well with their limited capacity, however it 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. 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. This also means that if VP9 can compress higher resolution frames than VP8 at the same target bitrate, we should pick VP9. Tab. 2 shows the resolution and codec we choose for different target bitrate ranges in our implementation.

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

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 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.

Encoding Video During Training. A key insight in the design of Gemino is that we need to design the neural compression pipeline to 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. 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 Super-Resolution 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.

5.5 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. 11 shows (in black) the target bitrate, along with the achieved bitrates of both schemes, and the associated perceptual quality [20] for a single video over the course of 220s of video-time[5]. We observe that initially (first 120s), at

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

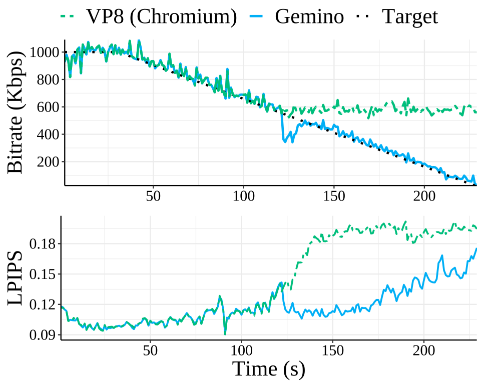


Figure 11: Gemino's ability to adapt to a time-varying target bitrate. As the target bitrate reduces, Gemino gradually lowers its PF stream resolution trading off more upsampling and less quality (increased LPIPS) for a reduction in achieved bitrate. VP8, in contrast, lowers the bitrate initially, but once at its minimum quality, it stops responding to the target bitrate.

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.

6 Limitations and Future Work

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, and not as widely supported on devices without access to some graphical processing engine. However, we believe that device improvements year on year are trending in a favorable direction, particularly with the emergence of optimized runtimes and hardware for running machine learning workloads on both Apple and Android devices [68,69]. Further, NetAdapt [18] and layer-by-layer pruning is only one technique amongst a large suite of model optimization approaches. We believe that with more targeted optimizations for particular devices, we can do better. Such optimizations become more salient when operating on higher-resolution video (*e.g.,* 4K, UltraHD) and in higher bandwidth regimes (∼ 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, sending more frequent reference frames 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.

7 Conclusion

This paper 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.

Acknowledgments

We thank our shepherd, Arpit Gupta, and our anonymous NSDI reviewers for their feedback. This work was supported by GIST, seed grants from the MIT Nano NCSOFT program and Zoom Video Communications, and NSF awards 2105819, 1751009 and 1910676.

References

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

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

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

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

[5] 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.

[6] 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.

[7] 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.

[8] 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.

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

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

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

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

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

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

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

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

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

[29] Pascal Getreuer. Linear Methods for Image Interpolation. *Image Processing On Line*, 1:238–259, 2011. https://doi.org/10.5201/ipol.2011.g_lmii.

[30] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Learning a deep convolutional network for image super-resolution. In *IEEE European Conference on Computer Vision (ECCV)*, 2014.

[31] Xintao Wang, Ke Yu, Shixiang Wu, Jinjin Gu, Yihao Liu, Chao Dong, Yu Qiao, and Chen Change Loy. Esrgan: Enhanced super-resolution generative adversarial networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 0–0, 2018.

[32] 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.

[33] 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.

[34] 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.

[35] 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.

[36] 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.

[37] 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.

[38] 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.

[39] 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.

[40] 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.

[41] 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.

[42] 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.

[43] 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.

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

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

[46] 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.

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

[48] 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.

[49] 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.

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

[51] WebRTC. https://webrtc.org/.

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

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

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

[55] 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.

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

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

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

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

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

[61] 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.

[62] 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.

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

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

[65] 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.

[66] 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.

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

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

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

[70] 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.

[71] 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.

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

[73] 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.

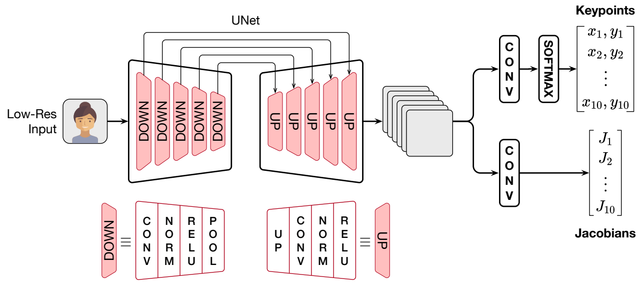


Figure 12: Keypoint Detector used as a precursor for computing the warping field between the reference and target images. Low-resolution versions of both frames are supplied to a UNet architecture [46], and then put through convolutional layers to generate keypoint locations and four "Jacobian" values in the neighborhood of each keypoint.

A 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. 12 and Fig. 13) to extract features from their respective inputs before they are post-processed. In both cases, the UNet consists of five up and down-sampling blocks each. Each downsampling block consists of a 2D convolutional layer [70], a batch normalization layer [71], a Rectified Linear Unit Non-linearity (ReLU) layer [72], 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 up-sampling 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 [5]. To extract keypoints, we first downsample the input image to 64×64, 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. 12 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. 13 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 in the neighborhood of each keypoint using the first-order Taylor series approximation [5] 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

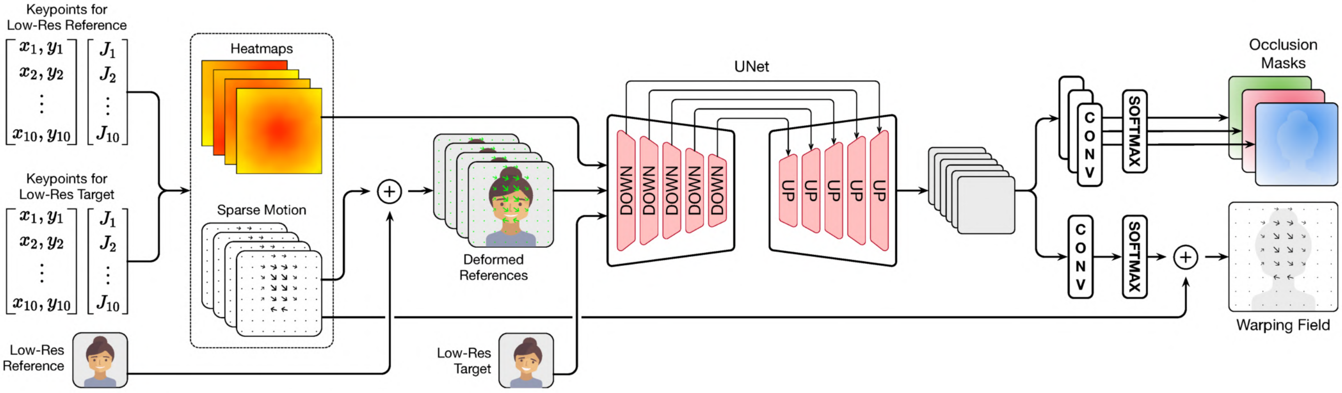


Figure 13: Gemino's motion estimation module that takes as input reference and target keypoints, along with a low-resolution reference and target frames, and produces a warping field and occlusion masks. The warping field helps the generator move encoded features from a full-resolution reference frame 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.

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.

A.2 Image Synthesis

Fig. 14 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. 12). 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 features (Ⓑ) are fed through five Residual Blocks [73] 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 [61], a feature-matching loss [62], and an L1 pixelwise 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 [17] with a weight of one (one-tenth of the other losses). The keypoints use an equivariance loss similar to the FOMM [5]. Our keypoint detector is unchanged relative to the FOMM, and so we reuse a trained checkpoint on the VoxCeleb dataset [45], 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. 8 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.

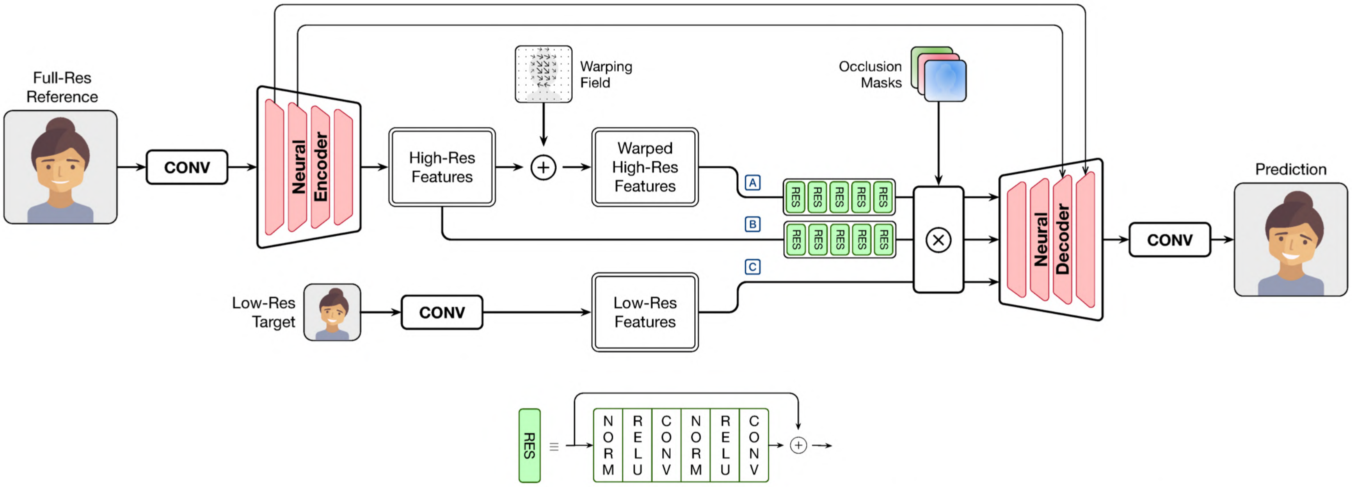


Figure 14: Gemino's encoder-decoder pair that is responsible for synthesizing the prediction. The encoder runs the high-resolution reference image through a series of downsampling layers to produce encoded features. A copy of these encoded HR features is warped, and both the warped and non-warped features 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.

	Training Videos		Test Videos	
Youtuber	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 8: Details of our dataset. All videos are at 1024×1024.

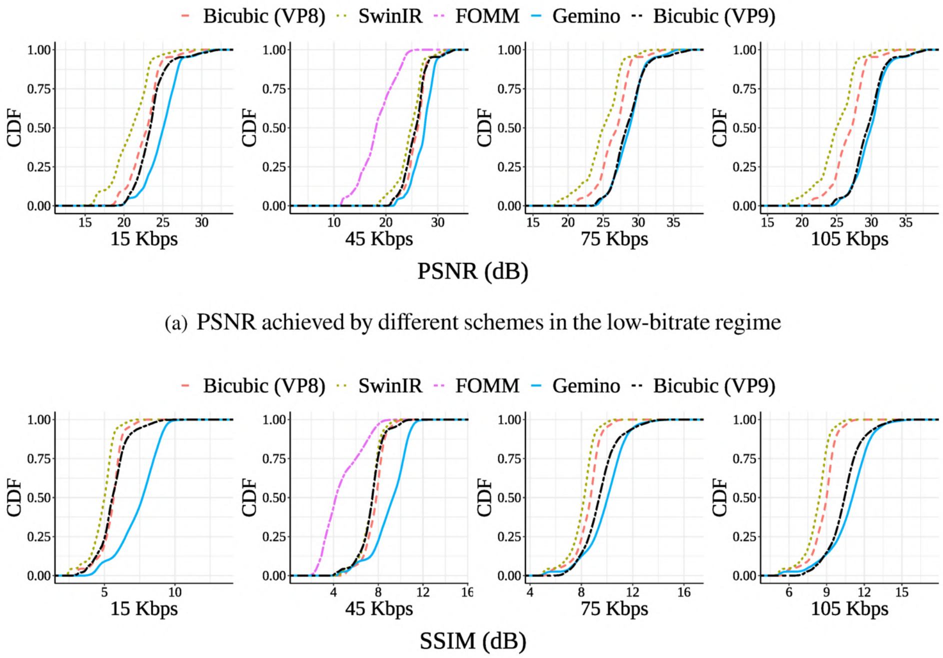
B Extended Evaluation

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. 15 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 §5.4 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 [20] primarily, and SSIM when relevant.

B.2 Comparing the metrics.

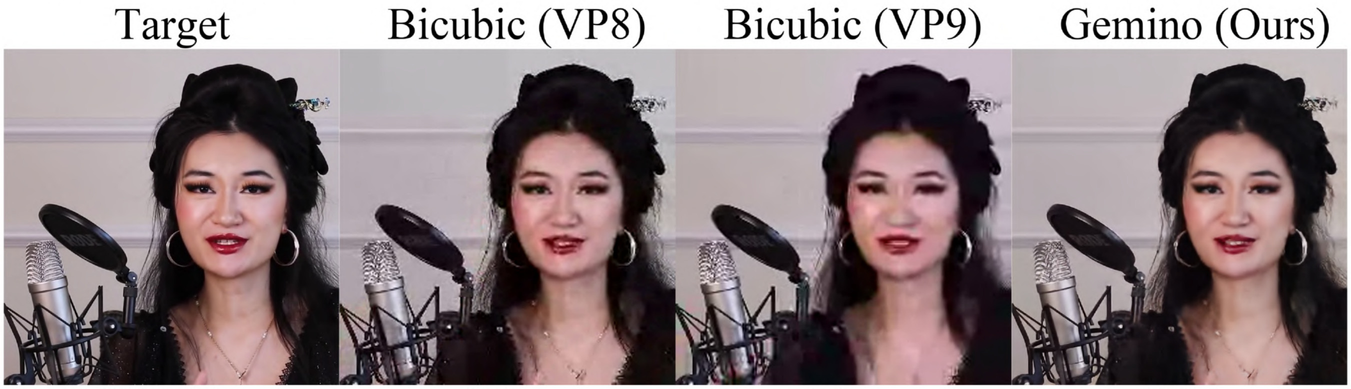
To understand the difference between different quantitative metrics and the visual quality each of them map to, we show a strip (with the associated metrics) that compares Gemino with the reconstruction of bicubic atop VP8 and VP9 when upsampling 256×256 frames at 45 Kbps in Fig. 16, 17, 18 and 19. For example, in Fig. 16, while the 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. 17 and in Fig. 18, where a difference of 0.13-0.14 in LPIPS, and ∼2 dB PSNR manifests as smoother output on the facial regions. In contrast, while the PSNRs in Fig. 19 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. 6 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. 17 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



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

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

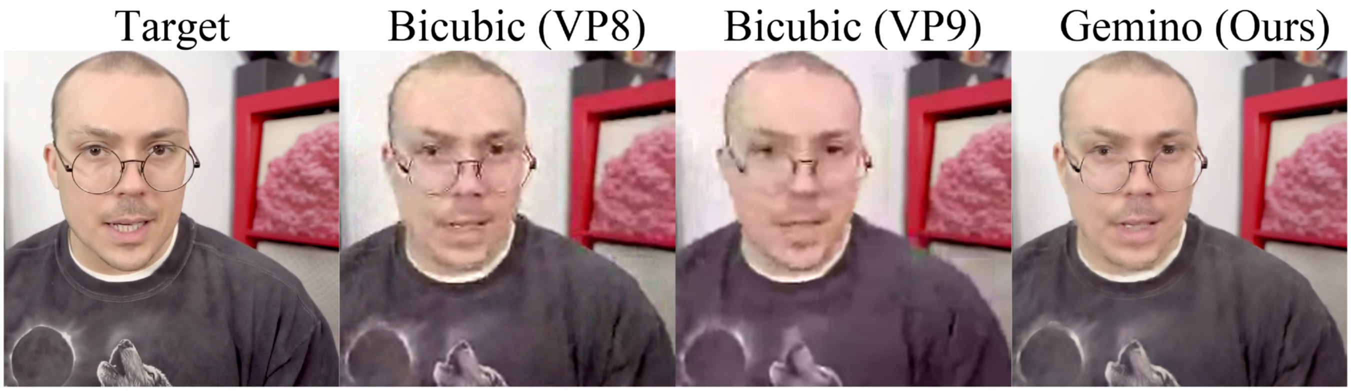
Figure 15: CDF of PSNR and SSIM achieved by different schemes on all frames in low-bitrate regimes. The differences between Gemino and other approaches becomes more pronounced in lower-bitrate regimes.



	Bicubic (VP8)	Bicubic (VP9)	Gemino (Ours)
PSNR (dB):	25.40	24.71	27.82
SSIM (dB):	7.81	7.42	10.10
LPIPS:	0.29	0.30	0.19

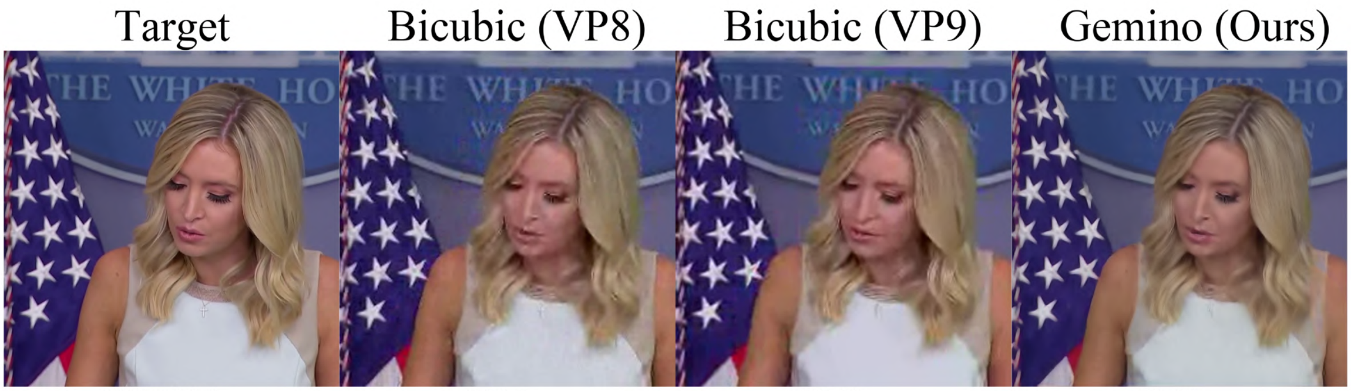
Figure 16: Full-size examples to illustrate the visual quality differences and their correlation with the metric values.

all along with visual strips is needed for a thorough evaluation.



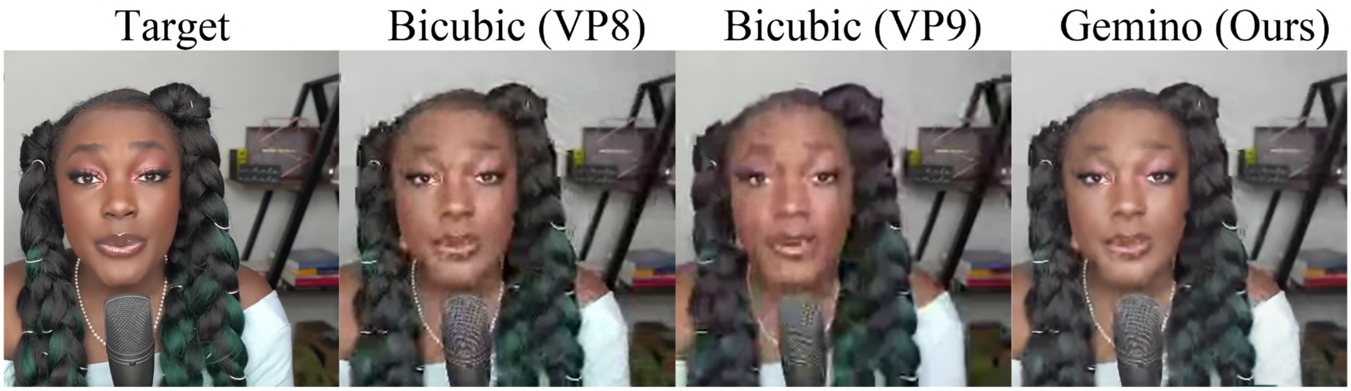
PSNR (dB):	26.80	26.40	28.85
SSIM (dB):	7.44	7.22	9.91
LPIPS:	0.34	0.33	0.20

Figure 17: 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 18: 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 19: Full-size examples to illustrate the visual quality differences and their correlation with the metric values.