Neural Video Compression using GANs for Detail Synthesis and Propagation

Fabian Mentzer*
Google Research
mentzer@google.com

Eirikur Agustsson*
Google Research
eirikur@google.com

Johannes Ballé
Google Research
jballe@google.com

David Minnen
Google Research
dminnen@google.com

Nick Johnston
Google Research
nickj@google.com

George Toderici
Google Research
gtoderici@google.com

Abstract

We present the first neural video compression method based on generative adversarial networks (GANs). Our approach significantly outperforms previous neural and non-neural video compression methods in a user study, setting a new state-of-the-art in visual quality for neural methods. We show that the GAN loss is crucial to obtain this high visual quality. Two components make the GAN loss effective: we i) synthesize detail by conditioning the generator on a latent extracted from the warped previous reconstruction to then ii) propagate this detail with high-quality flow. We find that user studies are required to compare methods, i.e., none of our quantitative metrics were able to predict all studies. We present the network design choices in detail, and ablate them with user studies.

1. Introduction

Recent work in neural *image* compression [2, 23], formalized in "rate-distortion-perception theory" [8,35,36,38], has shown how generative adversarial [12] image compression methods can outperform the HEVC-based image codec BPG [7] in terms of visual quality as measured by a user study, even when the neural approach uses half the bitrate. Interestingly, this gap in bitrate was not captured by the quantitative image quality metrics, and in fact, both PSNR and MS-SSIM predicted the opposite result.

At the time of writing, generative adversarial methods that target subjective quality for neural *video* compression remain unexplored. In general, the focus has been on outperforming the standard codecs in terms of PSNR or MS-SSIM with novel architectures or training schemes, leading to the latest approaches being comparable to or even outperforming HEVC [17] in terms of PSNR [1, 32, 45] or outperforming it in MS-SSIM [11, 32].

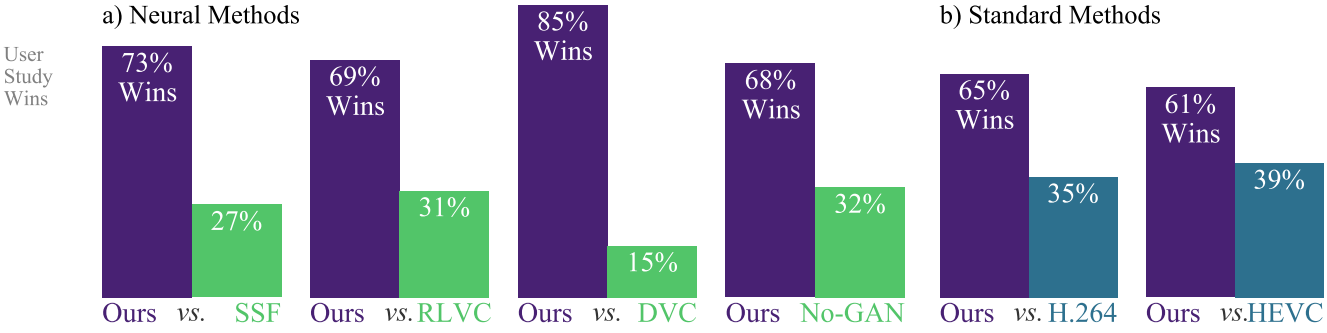
While the progress in image compression is promising, it is not obvious how to adapt the approach to the video domain. We need to be able to synthesize detail whenever new content appears, and then we need to propagate this detail to future frames. With this in mind, we carefully design a neural video compression architecture excelling at synthesizing and then preserving detail. Our approach works well when unrolled for many more frames than trained for: we train with up to 6 frames and show 60 frames in the user study. Our main contributions are as follows:

1. We present the first GAN-based neural compression system and set a new state-of-the-art in subjective visual quality measured with user studies, where we significantly outperform previous neural compression systems ([1], [44], [22]), as well as the standard codecs H.264 [4] and HEVC [17]. We show that the GAN loss is cruicial for this performance.
2. We show that two components are crucial to make the GAN loss effective: i) We condition the generator on a "free" (in terms of bits) latent obtained by feeding the warped previous reconstruction through the image encoder, and show that this is crucial to *synthesize* details. ii) To be able to *propagate* previously synthesized details, we rely on accurate optical flow provided by *UFlow* [18], and warping with high-quality resampling kernels.
3. We explore correlations between visual quality as measured by the user study and available video quality metrics, and show that *none of the metrics predict all studies*. To facilitate future research in this direction, we release reconstructions on the MCL-JCV data set along with all the data obtained from our user studies (links in App. B).

2. Related Work

Neural Video Compression Wu *et al*. [43] use frame interpolation for video compression, compressing B-frames by interpolating between other frames. Djelouah *et al*. [10] also use interpolation, but additionally employ an optical flow predictor for warping frames. This approach of using future frames is commonly referred to as "B-frame coding" for "bidirectional prediction". Other neural video coding methods rely on only using predictive (P) frames, commonly referred to as the "low-delay" setting, since it is more suitable for streaming applications by not relying on future frames. Lu *et al*. [22] use previously decoded frames and a

*Equal contributions.



Do metrics predict the user studies?

PSNR	*No*	*Yes*	*Yes*	*No*	*No*	*No*
MSSSIM	*No*	*No*	*Yes*	*No*	*No*	*No*
VMAF	*Yes*	*Yes*	*Yes*	*No*	*No*	*No*
PIM-1	*Yes*	*Yes*	*Yes*	*Yes*	*No*	*No*
LPIPS	*Yes*	*Yes*	*Yes*	*Yes*	*No*	*No*
FID/256	*Yes*	*Yes*	*Yes*	*Yes*	*No*	*No*

Figure 1. Comparing 6 different pairs of methods in a *user study*, on the 30 videos of MCL-JCV. We visualize how many user study comparisons each method won in the bar chart. We had 1639 ratings in total, with an average of 273 per method pair. Below each study, we answer the question: "Do metrics predict the user studies?" for each quantitative metric (underlying data shown in App. A.1). We see that none of the metrics predict all the studies, *i.e.*, no row is full of "Yes", showing the importance of user studies. *a)* Shows neural methods. We compare to the published methods *SSF* [1], *RLVC* [44], and *DVC* [22], seeing that our method significantly outperforms them. We compare *Ours* to our *no-GAN* baseline, where we see that a GAN clearly helps. *b)* We compare to the standard codecs H.264 [4] and HEVC [17], and see that our method also wins these comparisons. See the text for details.

pretrained optical flow network. Habibian *et al*. [15] do not explicitly model motion, and instead rely on a 3D autoregressive entropy model to capture spatial and temporal correlations. Liu *et al*. [20] build temporal priors via LSTMs, while Liu *et al*. [21] condition entropy models on previous frames. Rippel *et al*. [33] support adapting the rate during encoding, and also do not explicitly model motion. Agustsson *et al*. [1] propose "scale-space flow" to avoid complex residuals by allowing the model to blur as needed via a pyramid of blurred versions of the image. Yang *et al*. [45] generalize various approaches by learning to adapt the residual scale, and conditioning residual entropy models on flow latents. Golinsky *et al*. [11] recurrently connect decoders with subsequent unrolling steps, while Yang *et al*. [44] also add recurrent entropy models. Rippel and Anderson *et al*. [32] explore ways to make neural video compression more practical, with models that cover a range of bitrates and a focus on computational efficiency, improving encode and decode time.

Non-Neural Video Compression The combination of transform coding [14] using discrete cosine transforms [3] with spatial and/or temporal prediction, known as "Hybrid video coding", emerged in the 1980s as the technology dominating video compression until the present day. Non-neural methods such as H.261 through H.265/HEVC [17], VP8 [6], VP9 [28] and AV1 [9] have all remained faithful to the hybrid coding principle, with extensive refinements, regarding more flexible pixel formats (*e.g.*, bit depth, chroma subsampling), more flexible temporal and spatial prediction (*e.g.*, I-, P-, B-frames, intra block copy), and many more. Thanks to the years of research that went into these codecs, they provide strong baselines for neural approaches.

3. Method

3.1. Overview

An overview of the architecture we use is given in Fig. 2, while a detailed view with all layers is provided in App. Fig. 15. Let $x = \{x_1, x_2, \dots\}$ be a sequence of frames, where x_1 is the initial (I) frame, denoted by x_I in the figure and below. Similar to previous work, we operate in the "low-delay" mode, and hence predict subsequent (P) frames from previous frames. Let $\hat{x} = \{\hat{x}_1, \hat{x}_2, \dots\}$ be the reconstructed video. We use the following strategy to obtain high-fidelity reconstructions:

(S1) Synthesize plausible details in the I-frame.

(S2) Propagate those details wherever possible and as sharp as possible.

(S3) For new content appearing in P-frames, we again want to synthesize plausible details.

As mentioned in the Introduction, we optimize for perceptual quality and distortion, and note that the above three points are in contrast to purely distortion-optimized neural video codecs, which, particularly at low bitrates, favor blurring detail to reduce the distortion loss. To address these three points, we design our architecture as follows.

The **I-frame branch** is based on a lightweight version of the architecture used in HiFiC [23] (mostly making it less wide, see App. Fig. 15), and is used to address (S1). In detail, the encoder CNN E_I maps the input image x_I to

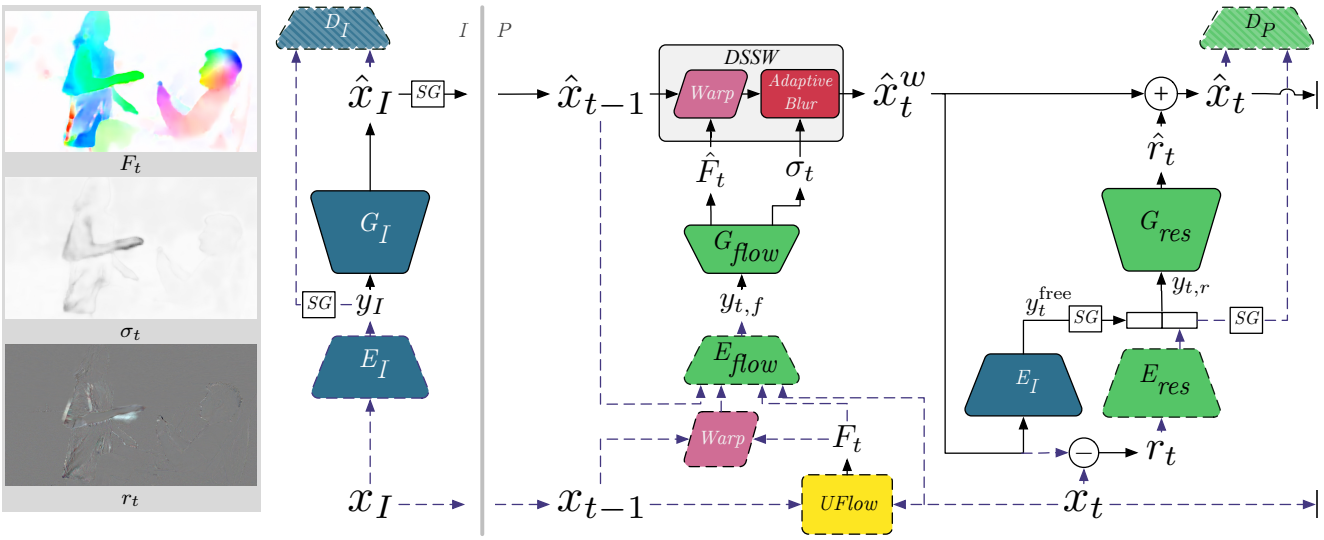


Figure 2. Architecture overview, with some intermediate tensors visualized in the gray box. To the *left* of the gray line is the I-frame branch (learned CNNs in blue), to the *right* the P-frame branch (learned CNNs in green). Dashed lines are not active during decoding, and discriminators D_I, D_P are only active during training. The size of the CNN blocks roughly indicates their capacity. SG is a stop gradient operation. *DSSW* is our "decoupled scale-space warping" (Sec. 3.2), and *UFlow* is a frozen optical flow model from [18].

a quantized latent y_I, which is entropy coded using a hyperprior [25] (not shown in Fig. 2, but which is detailed in App. Fig. 15). From the decoded y_I, we obtain a reconstruction $\hat{x}_I$ via the I-generator G_I. We use an I-frame discriminator D_I that—following [23]—is conditioned on the latent z_I (we elaborate on conditioning in Sec. 3.5).

The **P-frame branch** has two parts, an auto-encoder for the flow (E_{flow}, G_{flow} in Fig. 2), and an auto-encoder for the residual (E_{res}, G_{res}), following previous video work (*e.g.* [1, 22], *etc.*). To partially address (S2), similar to previous work, we employ a powerful optical flow predictor network on the encoder side, ***UFlow*** [18]. The resulting (backward) flow $F_t = UFlow(x_t, x_{t-1})$ is fed to the flow-encoder E_{flow}, which outputs the quantized and entropy-coded flow-latent $y_{t,f}$. From the flow-latent, the generator G_{flow} predicts both a reconstructed flow $\hat{F}_t$, as well as a mask σ_t. The mask σ_t has the same spatial dimensions as F_t, with each value in $[0, \sigma_{\text{max}}]$. Together, $(\hat{F}_t, \sigma_t)$ are used for our **decoupled scale-space warping**, a variant of *scale-space warping* [1], described in Sec. 3.2. Intuitively, for each pixel, the mask σ_t predicts how "correct" the flow at that pixel is (see the gray box in Fig. 2). We first warp the previous reconstruction $\hat{x}_{t-1}$ using $\hat{F}_t$, then we use σ_t to decide how much to blur each pixel. In practice, we observe σ_t predicts where new content appears which is not well captured by the flow and warping. Since the flow is in general relatively easy to compress, we employ shallow networks for E_{flow} and G_{flow} based on networks used in image compression [25]. We denote the resulting warped and potentially blurred previous reconstruction with $\hat{x}_t^w$.

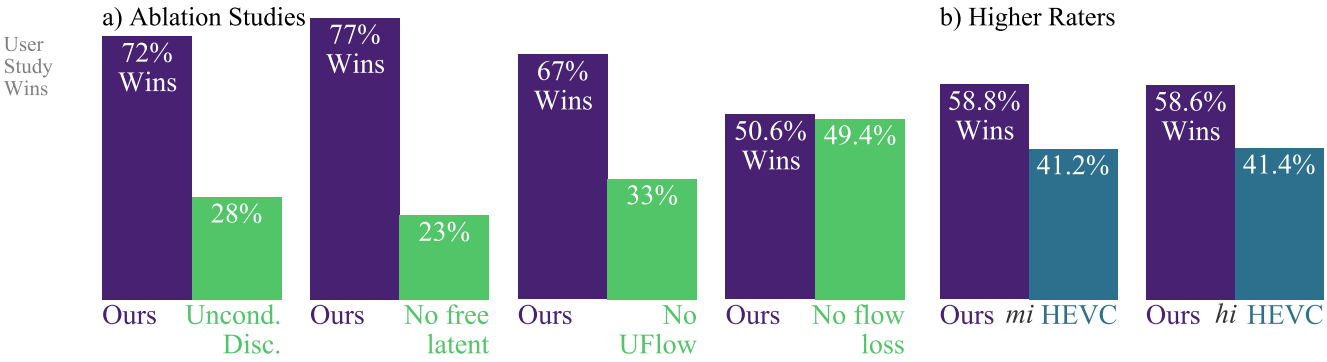
Finally, we calculate the residual $r_t = x_t - \hat{x}_t^w$ and compress it with the **residual auto-encoder** E_{res}, G_{res}. To address the last point above, (S3), we again employ the light version of the HiFiC architecture for E_{res}, G_{res}. However, we introduce one important component. We observe that G_{res} is not able to synthesize high-frequency details from the sparse residual latent $E_{res}(r_t)$ alone. However, we found that additionally feeding a **"free" latent** extracted from the warped previous reconstruction $y_t^{\text{free}} = E_I(\hat{x}_t^w)$ significantly increased the amount of synthesized detail, possibly due to the additional information and context provided by $\hat{x}_t^w$. Note that this latent does not need to be encoded into the bitstream because the decoder already has $\hat{x}_t^w$ and can compute y_t^{free} directly (hence it is "free"), and thus also does not need to be quantized. Instead, we concatenate it to $E_{res}(r_t)$ as a source of information, forming $y_{t,r}$=concat$(y_t^{\text{free}}, E_{res}(r_t))$.

To train the P-frame branch, we employ a seperate P-frame discriminator D_P, with the same architecture as D_I, conditioned on the full generator input $y_{t,r}$.

3.2. Decoupled Scale-Space Warping

When warping previously reconstructed frames, we want to preserve detail as much as possible (whether real or synthesized, per (S2) above). Previous neural video compression approaches have commonly used bi-linear warping [10, 22, 33, 44], or tri-linear scale-space warping (SSW) [1, 32, 45]. However it is known from signal processing theory (see e.g. Nehab *et al.* [29, Fig. 10.6 on p. 64]) that for repeated applications of re-sampling, the quality of the interpolation kernel is crucial to avoid low-pass filtering the signal and blurring out details, as visualized in Fig. 4.

Motivated by these observations, we were interested in implementing the more powerful *bicubic* warping in SSW,



Do metrics predict the user studies?

PSNR	*No*	*Yes*	*Yes*	*Yes*	*No*	*No*
MSSSIM	*No*	*No*	*No*	*Yes*	*No*	*No*
VMAF	*Yes*	*Yes*	*Yes*	*Yes*	*No*	*No*
PIM-1	*Yes*	*Yes*	*No*	*No*	*No*	*No*
LPIPS	*Yes*	*Yes*	*No*	*No*	*No*	*No*
FID/256	*Yes*	*Yes*	*No*	*No*	*No*	*No*

Figure 3. *a)* User study for Ablations. We see that using an unconditional discriminator or no free latent hurts performance. On the flow side, not using UFlow hurts, but the flow loss has no clear win. *b)* Comparing models at higher rates, targeting 0.14bpp (*mi*) and 0.22bpp (*hi*).

but found that this makes the implementation significantly more complex when combined with the 3-D indexing of scale-space warping. Instead, to be able to efficiently use bicubic warping (and arbitrary other warping operations), we propose a variant of scale-space warping [1], where we *decouple* the operation into two steps: plain warping, followed by spatially adaptive blurring. We can then use off-the-shelf warping implementations for the first part.

Both variants, at their core, use the scale-space flow field $(\hat{F}, \sigma)$, which generalizes optical flow $\hat{F}$ by also specifying a "scale" σ, such that we get a triplet $(u_{ij}, v_{ij}, \sigma_{ij})$ for each target pixel (i, j), where u_{ij}, v_{ij} are the flow coordinates, and σ_{ij} is the blurring scale to use. We recall the method from [1]: To compute a *scale-space warped* result

$$x_{\text{out}} = \text{SSW}(x, \hat{F}, \sigma), \tag{1}$$

Bilinear Resampling Kernel Bicubic Resampling Kernel

Figure 4. To avoid blurry results when repeatedly warping, the quality of the resampling kernel is crucial. Here, we compare shifting an images 20 times with a fixed flow of 0.5px to the left for bilinear and bicubic.

the source x is first repeatedly convolved with Gaussian blur kernels to obtain a "scale-space volume" with L levels,

$$V(x) = [x, x * G(s_1), \cdots, x * G(s_{L-1})], \tag{2}$$

where $G(s_i)$ is the Gaussian blur kernel with std. deviation s_i, and $\{s_1, \ldots, s_{L-1}\}$ are hyperparameters defining how blurry each level in the volume is. The three coordinates of the scale-space flow field $(u_{ij}, v_{ij}, \sigma_{ij})$ are then used to *jointly warp and blur* the source image, retrieving pixels via tri-linear interpolation from the scale-space volume.

We obtain a **Decoupled SSW (DSSW)** result by combining plain warping with spatially adaptive blurring (AB),

$$x'_{\text{out}} = \text{DSSW}(x, \hat{F}, \sigma) = \text{AB}(\text{Warp}(x, \hat{F}), \sigma), \tag{3}$$

where *Warp* is plain warping, and *AB* is *functionally* the same as scale-space warping (SSW) with a zero flow, *i.e.* $\text{AB}(y, \sigma) := \text{SSW}(y, 0, \sigma)$, but can be implemented with a few lines of code using simple multiplicative masks for each level in the scale-space volume to apply the 1-D linear interpolation for each pixel (see App. A.4) .

Together, bicubic warping and adaptive blurring help to propagate sharp detail when needed, while also facilitating smooth blurring when needed (*e.g.*, for focus changes in the video). See App. Fig. 12 for a visualization of how a given input and sigma field σ_t get blurred via scale-space blur.

We found that on a GPU, DSSW using an optimized warping implementation and our AB was $2-3\times$ faster than a naive SSW implementation. In App. A.4, we validate our implementation by training models for MSE, and showing that DSSW with bilinear warping obtains similar PSNR as SSW, and DSSW with bicubic warping yields a better model.

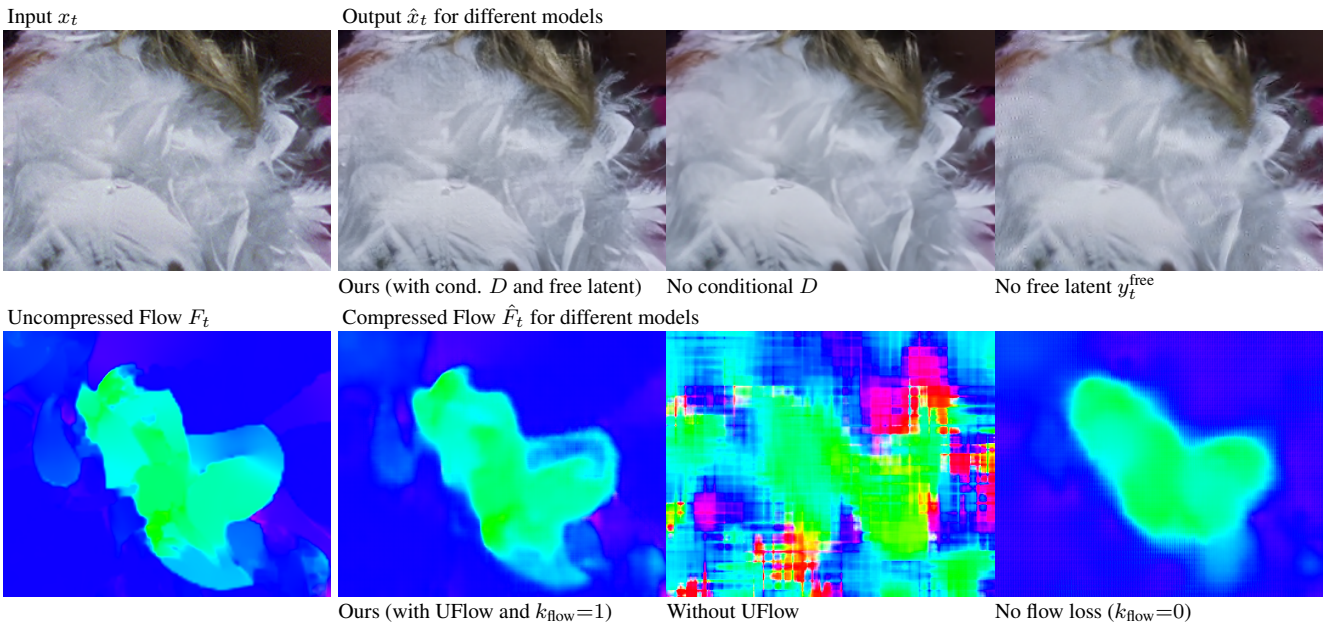


Figure 5. Visual examples for our ablations, see Sec. 5.1 for details. *Top*: Our model faithfully reconstructs details of the input, whereas making the discriminator D unconditional or removing the free latent introduces blurryness like in MSE models. *Bottom*: Not using supervised optical flow (*UFlow*) gives poor quality flows. Not using the flow loss makes the flow slightly burrier.

3.3. Adaptive Proportional Rate Control

We train our system by optimizing the rate-distortion-perception trade-off [8, 23], and we describe our formulation and loss in Sec. 3.5, but here we want to focus on one hyper-parameter in this trade-off (that also typically appears in the rate-distortion trade-off optimized by previous work): the weight on the bitrate, λ_R. It controls the trade-off between bitrate and other loss terms (distortion, GAN loss, *etc*.). Unfortunately, since there is no direct relationship between λ_R and the bitrate of the model when we vary other hyper-parameters, comparison across models is practically impossible, since they end up at different rates if we vary other hyper parameters, in particular other loss weights.

Van Rozendaal *et al*. [39] also observe this and propose targeting a fixed distortion via constraint optimization. Another approach was used in [23], where λ_R was dynamically selected from a small λ_1 and a large λ_2, depending on whether the model bitrate was below or above a given target. This approach can be interpreted as an "on-off" controller, but still some requires tuning of λ_1, λ_2.

A natural generalization is to use a **proportional-controller**: We measure the error between the current mini-batch bitrate b to a target bitrate b_t (in log-space), and apply it with a proportional controller to update λ_R as follows:

$$\log_2(\lambda_R) \leftarrow \log_2(\lambda_R) + k_P(\log(b+\epsilon) - \log(b_t+\epsilon)), \quad (4)$$

where $\epsilon = 1\text{E}{-9}$ for stability and the "proportional gain" k_P is a hyperparameter. We note that if we ignore the log-reparameterization, this corresponds to the "Basic Differential Multiplier Method" [31].

This approach is highly effective to obtain models that are comparable in terms of bitrate, despite different hyperparameters such as learning rates, amount of unrolling, loss weights, *etc*., as visualized in Fig. 6.

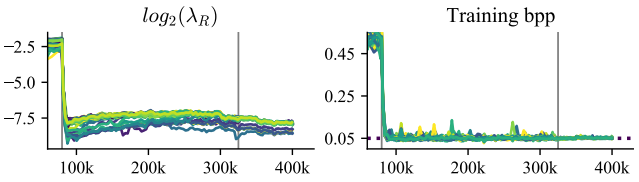


Figure 6. Visualizing the effect of the rate controller for a broad family of models with different hyper parameters, trained for 400k steps. The rate parameter λ_R is automatically adapted during training (*left*) to match the target bpp of 0.05 for all models (*right*). At 80k steps we drop the target rate, at 325k steps we drop the learning rate.

3.4. Sequence Length Train/Test Mismatch

One problem in neural video compression is the train/test mismatch in sequence length: Typically, neural approaches are trained on a handful of frames (*e.g*., three frames for [1] and five for [20]), and evaluated on hundreds of frames, which can lead to error patterns that emerge during evaluation. While the unrolling behavior is already a problem for MSE-optimized neural codecs (some previous works use small GOPs of 8-12 frames for evaluation to limit temporal error propagation), it requires even more care when detail is synthesized in a generative setting. Since we aim to

synthesize high-frequency detail whenever new content appears, incorrectly propagating that detail will create significant visual artifacts. Ideally, we could train with sequences as long as what we evaluate on (*i.e.*, at least T=60 frames), but in practice this is infeasible on current hardware due to memory and computational constraints. While we can fit up to T=12 into our accelerators, training then becomes prohibitively slow.

To work towards preventing unrolling issues, as well as accelerating prototyping and training new models, we instead adopt the following scheme: 1) Train E_I, G_I, D_I only, on randomly selected frames, for 1 000 000 steps. 2) Freeze E_I, G_I, D_I and initialize the weights of E_{res}, G_{res} from E_I, G_I. Train $E_{flow}, G_{flow}, E_{res}, G_{res}, D_P$ for 400 000 additional steps using **staged unrolling**, that is, use T=2 until $80k$ steps, T=3 until $300k$, T=4 until $350k$, and T=6 until $400k$. This splitting into steps 1) and 2) means trained E_I, G_I can be re-used for many variants of the P-frame branch, and, as a bonus, sharing E_I, G_I across models makes them more comparable. For training times, see App. A.8.

Some error accumulation remains, which we address in two ways: We quantize the frame buffer at each step, *i.e.*, during inference, we always quantize $\hat{x}_t$, to be closer to the (8-bit quantized) input. Additionally, we randomly shift reconstructions in each step, to avoid overlapping larger-scale error patterns from accumulating. We observe that together, these techniques help to get rid of most error patterns.

3.5. Formulation and Loss

We base our formulation on HiFiC [23] and optimize the rate-distortion-perception trade-off [8]. We use conditional GANs [12, 27], where both the generator and the discriminator have access to additional labels. As a short recap, the general conditional GAN formulation assumes data points x and labels s following some joint distribution $p(x, s)$. The generator is supposed to map samples $s \sim p(s)$ to the distribution $p(x|s)$, and the discriminator is supposed to predict whether a given pair (x, s) is from the real distribution $p(x|s)$, or from the generator distribution $p(\hat{x}|s)$.

In contrast to HiFiC, we are working with *sequences* of frames and reconstructions, however, we aim for per-frame distribution matching, *i.e.*, for T-length video sequences, the goal is to obtain a model s.t.:

$$p(\hat{x}_t|y_t) = p(x_t|y_t) \quad \forall t \in \{1, \dots, T\}, \tag{5}$$

where x_t are inputs, $\hat{x}_t$ reconstructions (as above), and we condition both the generators and the discriminators on latents y_t, using y_1=y_I for the I-frame, y_t=$y_{t,r}$ for P-frames (t>1). To readers more familiar with conditional generative video synthesis (*e.g.*, Wang *et al.* [41]), this simplification may seem sub-optimal as it may lead to temporal consistency issues (*i.e.*, you may imagine that reconstructions $\hat{x}_t, \hat{x}_{t+1}$ are inconsistent). We emphasize that since we are doing compression, we will also have a per-frame distortion loss (MSE), and we have information that we transmit to the decoder via a bitstream. So while the residual generator can in theory produce arbitrarily inconsistent reconstructions, in practice, these two points appear to be sufficient for preventing any temporal inconsistency issues in our models. We nevertheless explored variations where the discriminator is based on more frames, but this did not significantly alter reconstructions.

Continuing from Eq. 5, we define the overall loss for the I-frame branch and its discriminator D_I as follows. We use the "non-saturating" GAN loss [12]. To simplify notation, let $y_I = E_I(x_I), \hat{x}_I = G_I(y_I)$:[1]

$$\begin{aligned}\mathcal{L}_{\textit{I-Frame}} = \mathbb{E}_{x_I \sim p(x_I)}\big[\\ \lambda_R^I r(y_I) + d(x_I, \hat{x}_I) - \beta \log(D_I(\hat{x}_I, y_I))\big], \end{aligned} \tag{6}$$

$$\begin{aligned}\mathcal{L}_{D_I} = \mathbb{E}_{x_I \sim p(x_I)}\big[\\ -\log(1 - D_I(\hat{x}_I, y_I)) - \log(D_I(x_I, y_I))\big], \end{aligned} \tag{7}$$

where λ_R^I is the adaptive rate controller described in Sec. 3.3, β is the GAN loss weight, and d is a per-frame distortion. We use d=MSE, *i.e.*, in contrast to HiFiC [23], we *do not use a perceptual distortion* such as LPIPS. We found no benefit in training with LPIPS, possibly due to a more balanced hyper-parameter selection, and removing it speeds up training by ≈35%.

For the P-frame branch, let $p(x_1^T)$ be the distribution of T-length clips, where we use x_1 as the I-frame, and let

$$\begin{aligned}\mathcal{L}_{\textit{P-Frame}} = \mathbb{E}_{p(x_1^T)}\big[\textstyle\sum_{t=2}^{T} \lambda_R^P r(y_{t,r}) + t d(x_t, \hat{x}_t) \\ - t\beta \log(D_P(\hat{x}_t, y_{t,r})) + \mathcal{L}_{\text{reg}}\big], \end{aligned} \tag{8}$$

$$\begin{aligned}\mathcal{L}_{D_P} = \mathbb{E}_{p(x_1^T)}\big[\textstyle\sum_{t=2}^{T} -t \log(1 - D_P(\hat{x}_t, y_{t,r})) \\ - t \log(D_P(x_I, y_{t,r}))\big]. \end{aligned} \tag{9}$$

Note that we scale the losses of the t-th frame with t. This is motivated by the observation that $\hat{x}_t$ influences all $T - t$ reconstructions following it, and hence earlier frames indirectly have more influence on the overall loss. Scaling with t ensures all frames have similar influence.

Additionally, we employ a simple regularizer for the P-frame branch:

$$\mathcal{L}_{\text{reg}} = k_{\text{flow}} \cdot \texttt{SG}(\sigma_t) \cdot L_2(F_t, \hat{F}_t) + k_{\text{TV}} TV(\sigma_t), \tag{10}$$

where the first part is an MSE on the flow, ensuring that E_{flow}, G_{flow} learn to reproduce the flow from *UFlow*. We mask it with the sigma field, since we only require consistent flow where the network actually uses the flow (but add

[1] N.B.: Since we have deterministic auto-encoders, the distribution of $\hat{x}$ is fully defined via $p(x)$, and hence we only have expectations over $p(x_I)$, in contrast to typical GAN formulations.

Figure 7. User Study interface.

a stop gradient, SG, to avoid minimizing the loss by just predicting $\sigma_t = 0$). *TV* is a total-variation loss [34] ensuring a smooth sigma field.

4. Experiments

4.1. Datasets

Our **training** data contains 992k spatio-temporal crops of dimension 256×256, each containing 12 frames, obtained from videos from YouTube. For training, we randomly choose a contiguous sub-sequence of length $T \in \{2, 3, 4, 5\}$, see Sec. 3.4. The videos are filtered to originally be at least 1080p in resolution, 16:9, 30fps. We omit content labeled as "video games" or "computer generated graphics", using YouTube's category system [46].We **evaluate** our model on the 30 videos of MCL-JCV [40], which is available under a permissive license from USC, in contrast to, *e.g.*, the HEVC test sequences, which are not publicly available. MCL-JCV contains a broad variety of content types and difficulty, including a wide variety of motion from natural videos, computer animation and classical animation.

4.2. User Study

2AFC We evaluate our method in a user study, where we ask human raters to rate pairs of methods, *i.e.*, our setup is "two alternatives, forced choice" (2AFC). We implement 2AFC by showing raters two videos *side-by-side*, where the right video is always the *original*. On the left, raters see either a video from method A or method B. They can toggle between A and B in-place. We always shuffle the methods, *i.e.*, *Ours* is not always method A. We use all 30 videos from MCL-JCV, and show the first 2 seconds (to avoid large file sizes, see below), playing in a loop, but raters are allowed to pause videos. Raters are asked to "select the video that looks closest to the original". This protocol is inspired by previous work in image compression [23, 37], and ensures that differences between methods are easy to spot. The GUI is shown in Fig. 7, exact instructions in App. Fig. 8.

Several considerations went into these choices: For *generative* video compression, it is important to be able to compare to the original, as otherwise the method may, *e.g.*, completely change colors or content. However, we do not require pixel-perfect reconstructions, which is why we show the original on the right, and not in-place. Methods can be very similar, which is why we allow in-place switching *between methods* to be able to spot differences.

Rater Qualifications Our raters are contracted through the "Google Cloud AI Labeling Service" [13]. For each pair of methods, raters are asked to rate all 30 videos of MCL-JCV. In order to make sure our ratings have a high quality, we intersperse **five golden questions** at random locations into each study, where we compare HEVC at quality factor Q=27 to Q=35 (Q=27 yields bitrates similar to what we study, and Q=35 is $\approx$0.023bpp and contains significant blurring artifacts). We filter out raters who do not correctly answer 4 out of 5 of these questions. Overall, this yields 8-14 raters per study. To ensure that our results are *repeatable*, we repeat the study three days after first running it.

Shipping Videos to Raters In order to play back the videos in a web browser we transcode all methods with VP9 [28], using a very high quality factor to avoid any new artifacts. To ensure consistency, and to be sure that the raters can fit the task in their web browser, we center-crop the videos to 1080×1080. This yields large file sizes, so to ensure smooth playback, we *focus on the first 2 seconds (60 frames) of each video*.

4.3. Models and Metrics

We refer to our model as **Ours**, and report all hyperparameters in App. A.7. In contrast to most previous work, we do not constrain our model to use a small GoP size, and instead only use an I-frame for the first frame. To assess the effect of the GAN loss, we train a **no-GAN** baseline, which uses exactly the same architecture and training schedule as *Ours*, but is trained without a GAN loss (β=0 in Eqs. (6), (8)). We compare to three neural codecs: **SSF**, by Agustsson *et al.* [1], CVPR'2020, **RLVC** by Yang *et al.* [44], J-STSP'2021, and **DVC** by Lu *et al.* [22], CVPR'2019. For SSF and RLVC, we obtained reconstructions from the authors, for DVC we ran the open-sourced code ourselves with GOP=10 and verified that this does match their published numbers on the UVG dataset [24] (exact details in App. A.5). We ran user studies comparing all these models against our proposed GAN model. The neural codecs we compare to do not densely cover the bitrate axis, so to ensure fair studies, we fix *Ours* to a model targeting $\approx$0.07bpp, and then select a different competing model for each video to match filesizes as closely as possible. The resulting average bpps are at most $\approx3\%$ smaller or at most $\approx24\%$ bigger than our method. We emphasize that we would have liked to compare to even more neural models, but found no additional code or reconstructions.

Furthermore, we compare against the non-neural stan-

dard codecs **H.264** and **HEVC**. We follow best practice and make sure to minimally constrain the codecs, thus using the default "medium" preset (note that some previous works used "fast" or even "veryfast"). Like our method, we run the codecs in the low-latency setting (disabling B-frames). The exact (short) ffmpeg commands are listed in App. A.2. We also run the codecs at $\approx$0.07bpp, but to get an idea how models compare at higher rates, we additionally run *HEVC* at $\approx$0.14bpp and $\approx$0.22bpp.

In order to assess how quantitative metrics predict the user study, we employ the well-known PSNR and MS-SSIM [42]. We use LPIPS [47] (which measures a distance in AlexNet [19] features space) and PIM (an unsupervised quality metric), as well as FID [16]. Following HiFiC [23] we evaluate FID on non-overlapping 256×256 patches (see App. A.7 in [23]). Finally, we use VMAF [30], developed by Netflix to evaluate video codecs. We calculate these metrics on the exact sequences we ship to raters.

On Padding and Bitrates A problem faced by all CNN-based neural compression codecs is: what happens if the stride of the network does not divide the input resolution. For example, our encoder downscales 4 times, and thus needs the input resolution to divide 16. Like most previous work, we solve this by *padding* input frames (*e.g.*, 1080×1920 gets padded to 1088×1920), obtaining the bitstream of the padded image, obtaining the reconstruction, cropping the reconstruction back to the input resolution, and *calculating bpp w.r.t. the input resolution* (calculating it w.r.t. the padded resolution would amount to cheating). We note that the RLVC reconstructions were cropped to 1066 pixels, and we thus performed that user study in a cropped setting, and we had to add padding support to the DVC code, which may account for some differences in PSNR (DVC seems to have calculated on cropped images).

5. Results

Our results are shown in Fig. 1. Each pair of bars represents a pair of methods, and we show how many comparison each method won. At a high level, we see that *Ours* wins all user studies: Ours vs. no-GAN shows that the GAN loss significantly improves quality. The first three studies show that our method significantly outperforms all neural baselines. The standard codecs fare somewhat better, yet our method is clearly preferred overall. We show the comparison at higher rates in Fig. 3b, and we see that as we increase rates, the gap between methods start to get smaller. This makes sense given that at higher rates, both codecs look very good, and it becomes harder to distinguish them.

Below the bar chart, for each quantitative metric and each study, we show whether the metric predicts the study. The corresponding metric values are presented in App. A.1, *e.g.*, we can see there that we win against *no-GAN* in the user study, yet our method has 34.5dB PSNR, while *no-GAN* has 35.1dB (better), thus PSNR does *not* predict this study correctly, and we write "No". Overall, **none of the metrics are able to predict all studies**. However, we find that the three "perceptual" metrics PIM, LPIPS, and FID/256 all predict the studies of the *neural codecs*. Yet, they cannot be fully relied on, as none of them predicts the studies of the standard codecs.

In App. A.3, we show that we were able to obtain the same overall results when running the studies with the same raters three days later, with an even wider gap, and more raters passing the golden study. This indicates raters got somewhat familiar with the task. We also present statistics: how long raters take to answer questions, how often they flip, and how often they pause. We split this data by experiment, by video, and by worker. Averaged over all studies, raters take 26.4s per comparison, flip 13.5 times, and pause 0.967 times. To facilitate further research, we provide links to **reconstructions and raw user study data** in App. B.

5.1. Ablations

We ablate our main components, using a user study (shown in Fig. 3a) and visually (in Fig. 5). We do ablations by removing parts: In **Uncond. Disc.**, we train with an unconditional discriminator (*i.e.*, D does not see any latents), and the study shows that this performs significantly worse. **No free latent** shows that the free latent y_t^{free} is even more crucial. In Fig. 5 we see that removing either part yields significantly blurrier reconstructions. **No UFlow** shows that if we disable *UFlow*, *i.e.*, if we do not feed F_t to E_{flow}, and instead let E_{flow} learn flow unsupervised from frames, we perform significantly worse. In Fig. 5 we also see that the resulting flow looks very sub-optimal. Finally, **No flow loss** shows that removing the flow loss (*i.e.* setting $k_{\text{flow}}=0$ in Eq. 10) is actually preferable on some videos, but overall results in similar models, which aligns with what we see in Fig. 5, where the flow only changes on a small scale.

In summary, we see that a conditional discriminator, the free latent, and the accurate flow from *UFlow* are crucial for good visual quality.

6. Conclusion

We presented a GAN-based approach to neural video compression, that significantly outperforms previous neural and non-neural methods, as measured in a user study. With additional user studies, we showed that two components are crucial: i) conditioning the residual generator on a latent obtained from the warped previous reconstruction, and ii) leveraging accurate flow from an optical flow network. Furthermore, we showed how to decouple scale-space warping to be able to leverage high quality resampling kernels, and we used adaptive rate control to ensure consistent bitrates across a wide range of hyperparameters.

Societal Impact We note that our system can synthesize detail which is non-existent in the original video and hence its use is not appropriate for all types of content.

Limitations As we saw, the quantitative metrics we currently have cannot be fully relied on, and hence we have to do user studies. However, this is expensive and not very scalable, and further research into perceptual metrics is needed. We hope that by releasing our reconstructions, we can encourage research in this direction.

References

[1] Eirikur Agustsson, David Minnen, Nick Johnston, Johannes Balle, Sung Jin Hwang, and George Toderici. Scale-space flow for end-to-end optimized video compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8503–8512, 2020. 1, 2, 3, 4, 5, 7, 14, 17

[2] Eirikur Agustsson, Michael Tschannen, Fabian Mentzer, Radu Timofte, and Luc Van Gool. Generative adversarial networks for extreme learned image compression. In *The IEEE International Conference on Computer Vision (ICCV)*, October 2019. 1

[3] N. Ahmed, T. Natarajan, and K. R. Rao. Discrete cosine transform. *IEEE Trans. on Computers*, C-23(1), 1974. 2

[4] ITU-T rec. H.264 & ISO/IEC 14496-10 AVC: Advanced video coding for generic audiovisual services, 2003. 1, 2

[5] Johannes Ballé, David Minnen, Saurabh Singh, Sung Jin Hwang, and Nick Johnston. Variational image compression with a scale hyperprior. In *International Conference on Learning Representations (ICLR)*, 2018. 16

[6] 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. 2

[7] Fabrice Bellard. BPG Image format. https://bellard.org/bpg/. 1

[8] Yochai Blau and Tomer Michaeli. Rethinking lossy compression: The rate-distortion-perception tradeoff. *arXiv preprint arXiv:1901.07821*, 2019. 1, 5, 6

[9] 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. 2

[10] Abdelaziz Djelouah, Joaquim Campos, Simone Schaub-Meyer, and Christopher Schroers. Neural inter-frame compression for video coding. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6421–6429, 2019. 1, 3

[11] Adam Golinski, Reza Pourreza, Yang Yang, Guillaume Sautiere, and Taco S Cohen. Feedback recurrent autoencoder for video compression. In *Proceedings of the Asian Conference on Computer Vision*, 2020. 1, 2

[12] 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. 1, 6

[13] Google AI platform data labeling service. https://cloud.google.com/ai-platform/data-labeling/pricing. Accessed: 2021-05-01. 7

[14] Vivek K. Goyal. Theoretical foundations of transform coding. *IEEE Signal Processing Magazine*, 18(5), 2001. 2

[15] Amirhossein Habibian, Ties van Rozendaal, Jakub M Tomczak, and Taco S Cohen. Video compression with rate-distortion autoencoders. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7033–7042, 2019. 2

[16] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in Neural Information Processing Systems*, pages 6626–6637, 2017. 8

[17] ITU-T rec. H.265 & ISO/IEC 23008-2: High efficiency video coding, 2013. 1, 2

[18] Rico Jonschkowski, Austin Stone, Jonathan T Barron, Ariel Gordon, Kurt Konolige, and Anelia Angelova. What matters in unsupervised optical flow. *arXiv preprint arXiv:2006.04902*, 1(2):3, 2020. 1, 3, 18

[19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 8

[20] Haojie Liu, Tong Chen, Ming Lu, Qiu Shen, and Zhan Ma. Neural video compression using spatio-temporal priors. *arXiv preprint arXiv:1902.07383*, 2019. 2, 5

[21] Jerry Liu, Shenlong Wang, Wei-Chiu Ma, Meet Shah, Rui Hu, Pranaab Dhawan, and Raquel Urtasun. Conditional entropy coding for efficient video compression. *arXiv preprint arXiv:2008.09180*, 2020. 2

[22] Guo Lu, Wanli Ouyang, Dong Xu, Xiaoyun Zhang, Chunlei Cai, and Zhiyong Gao. Dvc: An end-to-end deep video compression framework. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11006–11015, 2019. 1, 2, 3, 7

[23] Fabian Mentzer, George D Toderici, Michael Tschannen, and Eirikur Agustsson. High-fidelity generative image compression. *Advances in Neural Information Processing Systems*, 33, 2020. 1, 2, 3, 5, 6, 7, 8, 16, 18

[24] Alexandre Mercat, Marko Viitanen, and Jarno Vanne. UVG dataset: 50/120fps 4K sequences for video codec analysis and development. In *Proceedings of the 11th ACM Multimedia Systems Conference*, pages 297–302, 2020. 7

[25] David Minnen, Johannes Ballé, and George D Toderici. Joint autoregressive and hierarchical priors for learned image compression. In *Advances in Neural Information Processing Systems*, pages 10771–10780, 2018. 3

[26] David Minnen and Saurabh Singh. Channel-wise autoregressive entropy models for learned image compression. *arXiv preprint arXiv:2007.08739*, 2020. 16

[27] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014. 6

[28] Debargha Mukherjee, Jim Bankoski, Adrian Grange, Jingning Han, John Koleszar, Paul Wilkins, Yaowu Xu, and Ronald Bultje. The latest open-source video codec vp9-an overview and preliminary results. In *2013 Picture Coding Symposium (PCS)*, pages 390–393. IEEE, 2013. 2, 7

[29] Diego Nehab, Hugues Hoppe, et al. *A fresh look at generalized sampling*. Citeseer, 2014. 3

[30] Netflix. VMAF - Video Multi-Method Assessment Fusion. https://github.com/Netflix/vmaf/. 8

[31] John C Platt and Alan H Barr. Constrained differential optimization for neural networks. *Caltech*, 1988. 5

[32] Oren Rippel, Alexander G Anderson, Kedar Tatwawadi, Sanjay Nair, Craig Lytle, and Lubomir Bourdev. ELF-VC: Efficient Learned Flexible-Rate Video Coding. *arXiv preprint arXiv:2104.14335*, 2021. 1, 2, 3

[33] Oren Rippel, Sanjay Nair, Carissa Lew, Steve Branson, Alexander G Anderson, and Lubomir Bourdev. Learned video compression. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3454–3463, 2019. 2, 3

[34] David Shulman and Jean-Yves Herve. Regularization of discontinuous flow fields. In *Proc. workshop on visual motion*, pages 81–86. IEEE Computer Society Press, 1989. 7

[35] Lucas Theis and Eirikur Agustsson. On the advantages of stochastic encoders. *arXiv preprint arXiv:2102.09270*, 2021. 1

[36] Lucas Theis and Aaron B Wagner. A coding theorem for the rate-distortion-perception function. *arXiv preprint arXiv:2104.13662*, 2021. 1

[37] George Toderici, Lucas Theis, Nick Johnston, Eirikur Agustsson, Fabian Mentzer, Johannes Ballé, Wenzhe Shi, and Radu Timofte. *CLIC 2020: Challenge on Learned Image Compression*, 2020. http://compression.cc. 7

[38] Michael Tschannen, Eirikur Agustsson, and Mario Lucic. Deep generative models for distribution-preserving lossy compression. In *Advances in Neural Information Processing Systems*, pages 5929–5940, 2018. 1

[39] Ties van Rozendaal, Guillaume Sautiere, and Taco S Cohen. Lossy compression with distortion constrained optimization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 166–167, 2020. 5

[40] Haiqiang Wang, Weihao Gan, Sudeng Hu, Joe Yuchieh Lin, Lina Jin, Longguang Song, Ping Wang, Ioannis Katsavounidis, Anne Aaron, and C-C Jay Kuo. MCL-JCV: a JND-based H.264/AVC video quality assessment dataset. In *2016 IEEE International Conference on Image Processing (ICIP)*, pages 1509–1513. IEEE, 2016. 7

[41] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Guilin Liu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. Video-to-video synthesis. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018. 6

[42] Zhou Wang, Eero P Simoncelli, and Alan C Bovik. Multiscale structural similarity for image quality assessment. In *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003*, volume 2, pages 1398–1402. Ieee, 2003. 8

[43] Chao-Yuan Wu, Nayan Singhal, and Philipp Krahenbuhl. Video compression through image interpolation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 416–431, 2018. 1

[44] Ren Yang, Fabian Mentzer, Luc Van Gool, and Radu Timofte. Learning for video compression with recurrent auto-encoder and recurrent probability model. *IEEE Journal of Selected Topics in Signal Processing*, 2020. 1, 2, 3, 7, 17

[45] Ruihan Yang, Yibo Yang, Joseph Marino, and Stephan Mandt. Hierarchical autoregressive modeling for neural video compression. *arXiv preprint arXiv:2010.10258*, 2020. 1, 2, 3

[46] *YouTube Data API*, 2021. https://www.googleapis.com/youtube/v3/videoCategories. 7

[47] 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. 8

A. Appendix

A.1. User Study Data

In Tables 1, 2, we show the data underlying the *Yes*/*No* results shown in the main paper user study figures (Figs. 1, 3).

	Ours	SSF	Predicts Study	Ours	RLVC	Predicts Study	Ours	DVC	Predicts Study	Ours	No-GAN	Predicts Study	Ours	H.264	Predicts Study	Ours	HEVC	Predicts Study
Study Wins↑	73%	27%		69%	31%		85%	15%		68%	32%		65%	35%		61%	39%	
PSNR↑	34.5	34.8	*No*$_{\approx}$	34.5	34	*Yes*	34.5	31.7	*Yes*	34.5	35.1	*No*	34.5	34.6	*No*$_{\approx}$	34.5	35.6	*No*
MS-SSIM↑	0.964	0.963	*No*$_{\approx}$	0.969	0.965	*No*$_{\approx}$	0.964	0.95	*Yes*	0.964	0.967	*No*$_{\approx}$	0.964	0.963	*No*$_{\approx}$	0.964	0.966	*No*$_{\approx}$
VMAF↑	87.3	84.8	*Yes*	87.3	83.1	*Yes*	87.3	81.9	*Yes*	87.3	86.9	*No*$_{\approx}$	87.3	87.7	*No*$_{\approx}$	87.3	91.1	*No*
PIM-1↓	3.34	4.69	*Yes*	3.35	4.93	*Yes*	3.34	6.91	*Yes*	3.34	4.17	*Yes*	3.34	3.17	*No*	3.34	2.62	*No*
LPIPS↓	0.168	0.224	*Yes*	0.168	0.224	*Yes*	0.168	0.26	*Yes*	0.168	0.194	*Yes*	0.168	0.169	*No*$_{\approx}$	0.168	0.147	*No*
FID/256↓	32.8	54.1	*Yes*	32.6	50.3	*Yes*	32.8	61.6	*Yes*	32.8	35.7	*Yes*	32.8	33	*No*$_{\approx}$	32.8	24.2	*No*

Table 1. We show the data underlying the *Yes/No* answers in Fig. 1. The first row shows the *user study results* (*i.e.*, what is represented with bars in Fig. 1). Subsequent rows indicate for each metric and each method whether the metric predicts the study, using *Yes* when the value for *Ours* is better than for the other method, *i.e.*, the metric *predicts* the study, and *No* when the metric *does not predict* the study. If the values are within 1% of each other, the metric also *does not predict* the study, and we indicate this with *No*$_{\approx}$. ↑ indicates that higher is better for this row, ↓ the opposite.

	Ours	Uncond. Disc.	Predicts Study	Ours	No free latent	Predicts Study	Ours	No UFlow	Predicts Study	Ours	No flow loss	Predicts Study	Ours, medium	HEVC, medium	Predicts Study	Ours, high	HEVC, high	Predicts Study
Study Wins↑	72%	28%		77%	23%		67%	33%		50.6%	49.4%		58.8%	41.2%		58.6%	41.4%	
PSNR↑	34.5	34.8	*No*$_{\approx}$	34.5	33.9	*Yes*	34.5	33.9	*Yes*	34.5	34.3	*Yes*$_{\approx}$	35.9	37	*No*	36.8	38.2	*No*
MS-SSIM↑	0.964	0.966	*No*$_{\approx}$	0.964	0.959	*No*$_{\approx}$	0.964	0.96	*No*$_{\approx}$	0.964	0.963	*Yes*$_{\approx}$	0.973	0.974	*No*$_{\approx}$	0.978	0.979	*No*$_{\approx}$
VMAF↑	87.3	85.6	*Yes*	87.3	81.9	*Yes*	87.3	84.2	*Yes*	87.3	86.9	*Yes*$_{\approx}$	93	94.7	*No*	94.9	96.5	*No*
PIM-1↓	3.34	3.83	*Yes*	3.34	3.85	*Yes*	3.34	3.32	*No*$_{\approx}$	3.34	3.29	*No*	2.73	2.15	*No*	2.07	1.75	*No*
LPIPS↓	0.168	0.172	*Yes*	0.168	0.194	*Yes*	0.168	0.167	*No*$_{\approx}$	0.168	0.161	*No*	0.114	0.112	*No*	0.0916	0.0895	*No*
FID/256↓	32.8	34.9	*Yes*	32.8	35.9	*Yes*	32.8	32.7	*No*$_{\approx}$	32.8	28.8	*No*	18.6	15.5	*No*	15.2	10.7	*No*

Table 2. We show the data underlying the *Yes/No* answers in Fig. 3. The first row shows the *user study results* (*i.e.*, what is represented with bars in Fig. 3). Subsequent rows indicate for each metric and each method whether the metric predicts the study, using *Yes* when the value for *Ours* is better than for the other method, *i.e.*, the metric *predicts* the study, and *No* when the metric *does not predict* the study. If the values are within 1% of each other, and we win the study by a significant margin (*i.e.*, the metric also *does not predict* the study), we indicate this with *No*$_{\approx}$. For the *No flow loss* ablation, the user study indicates that the methods perform similarly. Thus, if the metric values are within 1% of each other, the metric *does predict* the study, and we indicate this with *Yes*$_{\approx}$. ↑ indicates that higher is better for this row, ↓ the opposite.

A.2. HEVC/H.264 commands

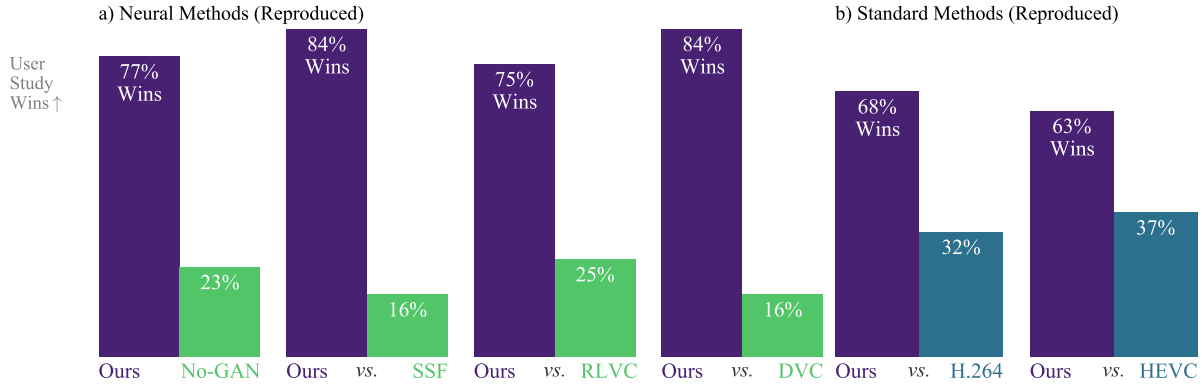
We use `ffmpeg`, v4.3.2, to encode videos with HEVC and H264. We use the `medium` preset, and no B-frames, as mentioned in Sec 4.3. We compress each video using quality factors `$Q` $\in \{10, \ldots, 35\}$ to find bpps that match our models, using the following commands:

```
ffmpeg -i $INPUT_FILE -c:v h264 \
    -crf $Q -preset medium \
    -bf 0 $OUTPUT_FILE

ffmpeg -i $INPUT_FILE -c:v hevc \
    -crf $Q -preset medium \
    -x265-params bframes=0 $OUTPUT_FILE
```

A.3. User Study Details

- The result of repeating the user studies three days later is shown in Fig. 8 (top).
- Rater instructions are shown in Fig. 8 (bottom).
- We show user study statistics in Fig. 9, see caption for details.



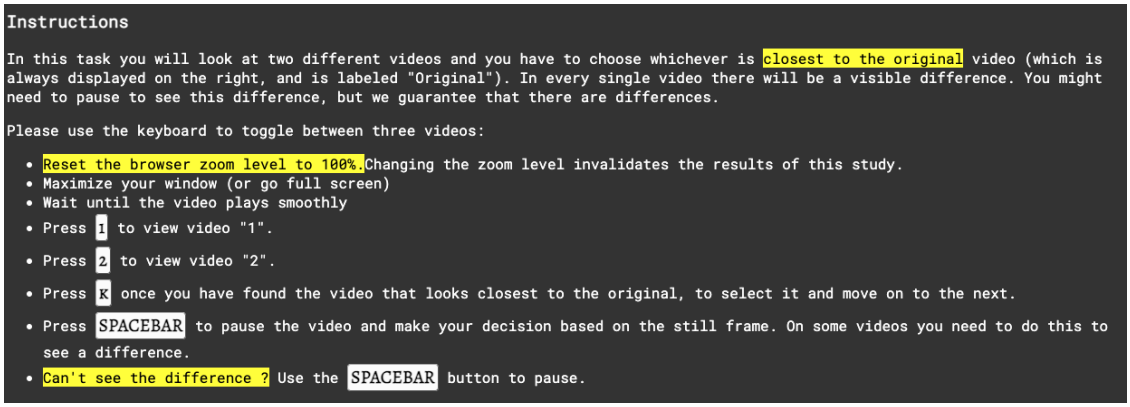


Figure 8. *Top:* Study repeated 3 days later. *Bottom:* Instructions shown to the raters.

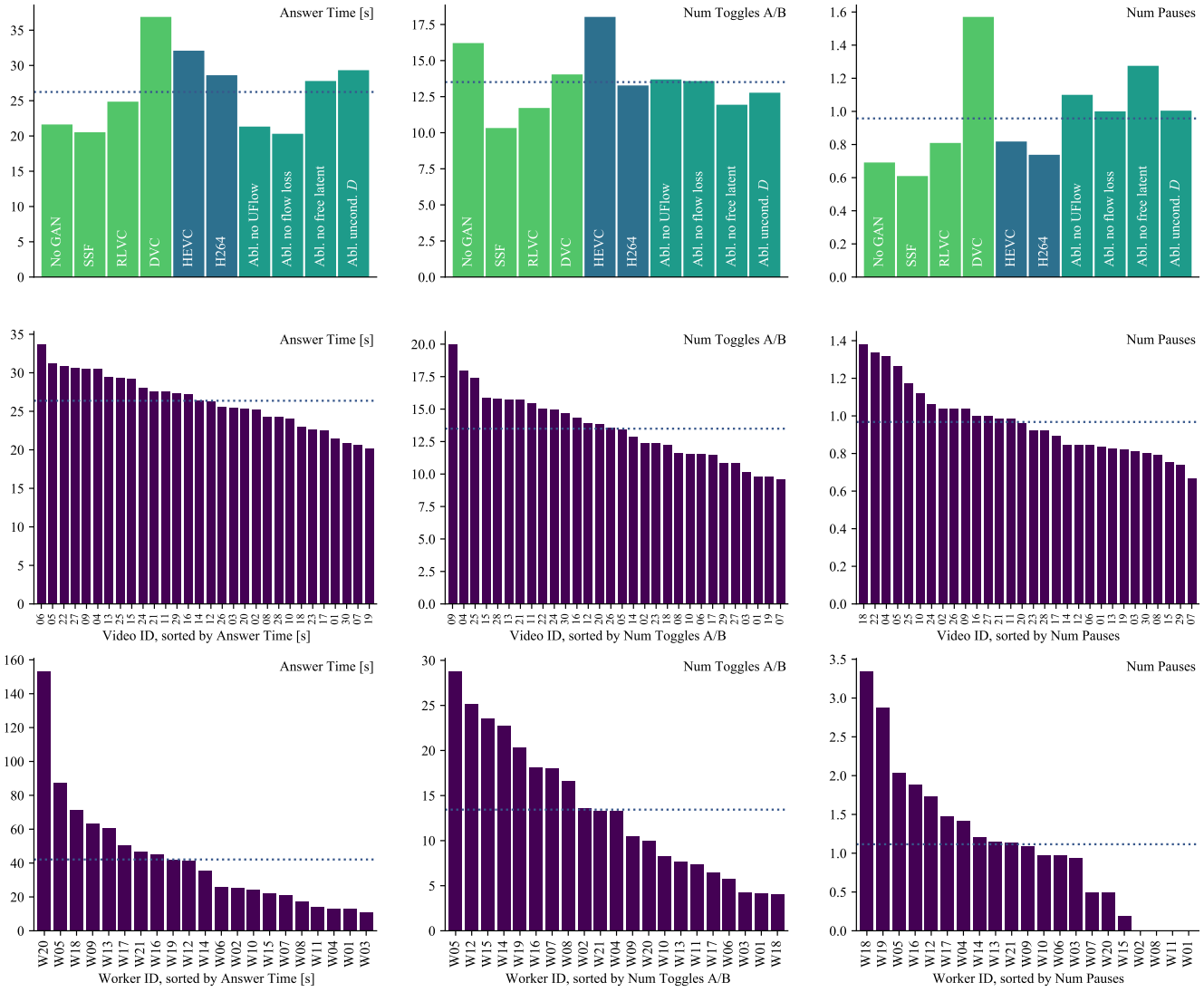


Figure 9. User study statistics, grouped in various ways. The dotted line in each plot indicates the mean of the values shown in the plot. From left to right, we show different statistics: Answer time in [s]econds, number of flips, and number of pauses. The rows show different ways of grouping the data: *Top*: We group by study, and color based on whether the study compares to a *neural codec*, to a *standard codec*, or is an *ablation*. *Middle*: We group by MCL-JCV video ID (01 to 30), and sort each plot. *Bottom*: We group by rater ID, and sort each plot (note that the means here are slightly different, as not all raters rate did the same number of studies).

```python
DEFAULT_SIGMAS = (0.0, 1.5, 3.0, 6.0, 12.0, 24.0)

def adaptive_blur(image: np.ndarray,
                  sigma_field: np.ndarray,
                  sigmas: Sequence[float] = DEFAULT_SIGMAS):
  """Blur `image` with scale field `sigma_field`.

  Args:
    image: A (B, H, W, 3) tensor.
    sigma_field: A (B, H, W, 1) tensor, representing sigma_t.
    sigmas: A list of L sigmas to use for the L levels in the
      scale space volume.

  Returns:
    A (B, H, W, 3) adaptively blurred image.
  """
  num_levels = len(sigmas)
  scale_space_volume = [
    gaussian_blur(image, sigma) for sigma in sigmas]

  # Our desired result is obtained by computing a 1-D
  # interpolation in the scale space volume for each pixel.
  # We collect the interpolation coefficients into `coeffs_by_level`,
  # which stores a (B, H, W, 1) matrix for each level.
  coeffs_by_level = [0.0 for _ in range(num_levels)]

  # Short-hand alias.
  w = sigma_field
  for level in range(num_levels - 1):
    s1 = sigmas[level]
    s2 = sigmas[level + 1]
    mask = ((w >= s1) & (w < s2)).astype(np.float32)
    # Now `mask` is a (B, H, W, 1) boolean mask indicating
    # all pixels which have a target sigma between `s1` and `s2`.
    # To find the appropriate interpolation coefficients
    # for those pixels, we follow the re-parameterization
    # in Sec. 3.1 in (Agustsson et al. 2020), which gives:
    t = (w**2 - s1**2) / (s2**2 - s1**2)
    coeffs_by_level[level] += (1 - t) * mask
    coeffs_by_level[level + 1] += t * mask

  # Return the interpolated result.
  return sum(
      coeffs_by_level[level] * scale_space_volume[level]
      for level in range(num_levels)
```

Figure 10. Numpy implementation of adaptive blurring.

A.4. Decoupled Scale-Space Warping: Details

We show a numpy version of adaptive blurring in Fig. 10, and a visualization of some variables in Fig. 12. To validate our implementation of decoupled scale-space warping (DSSW), we compare MSE-trained models in Fig. 11. We show that DSSW with bilinear warping is similar to scale-space warping with trilinear interpolation, validating our version. We see that using a bicubic resampling kernel, R-D performance improves by ≈6%. As we mention in Sec. 3.2, DSSW is also significantly faster to run on GPU. For the Figure, we used the architecture of [1] trained for MSE only, with a slightly accelerated training schedule by skipping the last training stage on larger (384px) crops, instead decaying the learning rate by 10x after 800 000 steps.

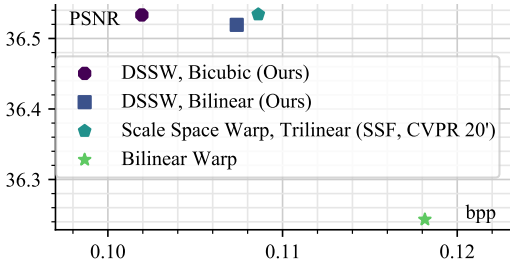
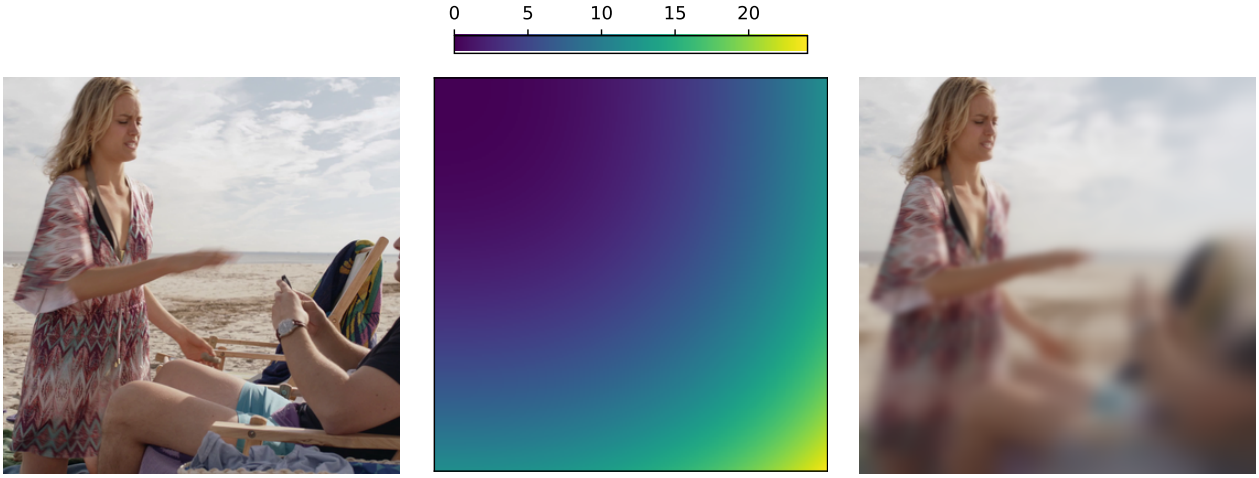
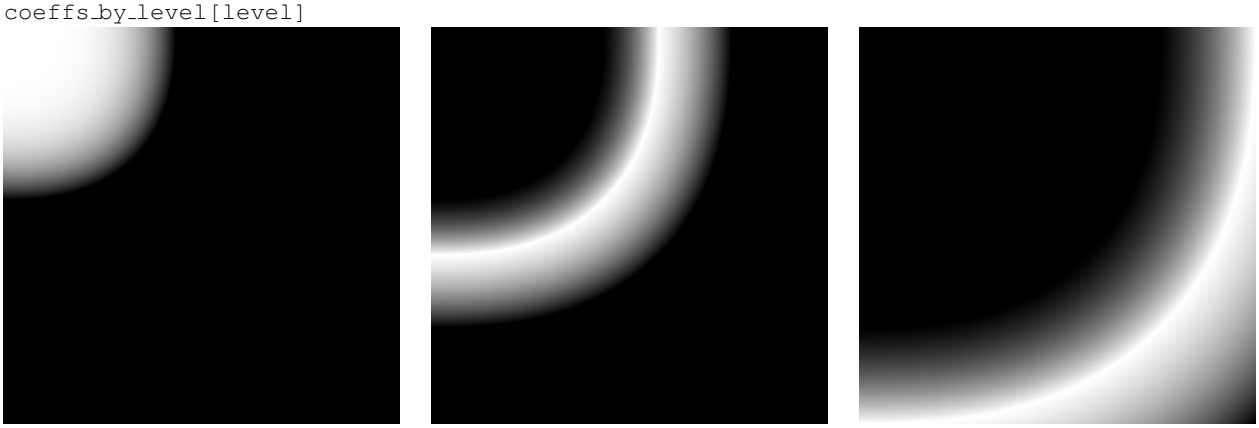


Figure 11. To validate our Decoupled Scale Space Warping (DSSW) implementation, we train models **for MSE**. We compare the R-D performance of Bilinear/Bicubic resamplers in DSSW against the Scale-Space Warping of Agustsson *et al.* [1] and find that DSSW with bicubic improves the bpp. We also show plain bilinear warping, without any scale space blurring.



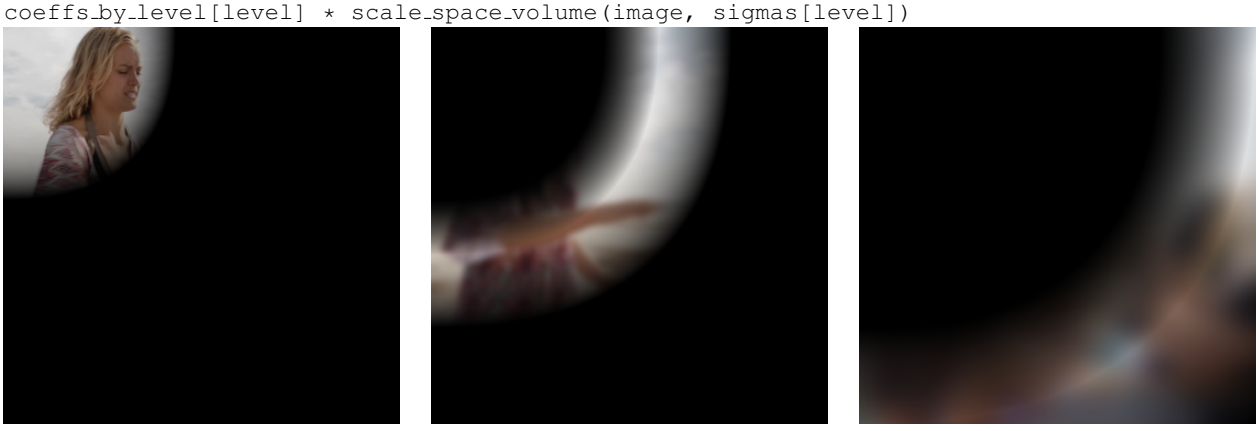
`image` | `sigma_field` | `adaptive_blur(image, sigma_field)`

`coeffs_by_level[level]`



`level = 0` | `level = 2` | `level = 4`

`coeffs_by_level[level] * scale_space_volume(image, sigmas[level])`



`level = 0` | `level = 2` | `level = 4`

Figure 12. Visualizing variables of the algorithm given in Fig. 10.

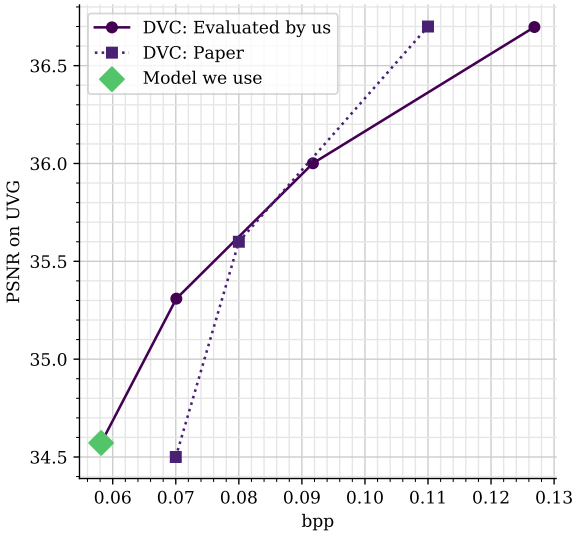


Figure 13. Comparing our DVC models to what the authors reported, on UVG. We use the model in the lower right, as this is closest to our bpps (achieving ≈ 0.06bpp on UVG, ≈ 0.09bpp on MCL-JCV).

A.5. DVC Details

To get DVC reconstructions, we use the code provided by the authors.[2] DVC uses the image compression model by Ballé *et al*. [5] for I-frames, but the code does not include the exact model. We thus tried all models, and picked the one with highest R-D performance, which is available as "bmshj2018-hyperprior-mse-5" in TFC.[3]. We note that we add padding and cropping as described in Sec. 4.3. We show the PSNR of our model obtained on UVG in Fig. 13.

A.6. Architecture Details

A detailed version of the architecture from Fig. 2 is given in Fig. 15.

A.7. Hyper Parameters

For scale space blurring we set $\sigma_0 = 1.5$ and used $L = 6$ levels, which implies that the sequence of blur kernel sizes is $[0.0, 1.5, 3.0, 6.0, 12.0, 24.0]$.

For rate control we initially swept over a wide range $k_P \in \{10^i, i \in \{-1, \dots, -9\}\}$ and found that 10^{-3} worked well, which we then fixed for all future experiments. We initialized $\log_2 \lambda_R = 1.0$ in all cases.

Previous works [23, 26] typically initially train for a higher bitrate. This is usually implemented by using a schedule on the R-D weight λ that is decayed by a factor $2\times$ or $10\times$ early in training. Since the rate-controllor automatically controls this weight, we emulate the approach by instead using a schedule on the targeted bitrate b_t. We use a simple rule and target a $+0.5$ higher bitrate for the first 20% of training steps.

For the I-frame loss $\mathcal{L}_{\textit{I-Frame}}$ (Eq. 6), we use $\beta = 128$, and $b_t = 0.4$ for rate-control.

For the P-frame loss $\mathcal{L}_{\textit{P-Frame}}$ (Eq. 8), we use $\beta = 128$, and $k_{\text{TV}} = 10.0, k_{\text{flow}} = 1.0$ in $\mathcal{L}_{\text{reg}}$. For the three different models we use in the user study, we use $b_t \in \{0.05, 0.10, 0.15\}$. A detail omitted from the equation is that we scale the loss by the constant $C_T = 1/T \sum_{t=2}^{T} t$, as this yields similar magnitudes as no loss scaling.

We use the same learning rate LR $= 1\text{E}{-4}$ for all networks, and train with the Adam optimizer. We linearly increase the LR from 0 during the first $20k$ steps, and then drop it to LR $= 1\text{E}{-5}$ after $320k$ steps. We train the discriminators for 1 step for each generator training step.

A.8. Training Time

In Table 3 we report the training speed for each of the training stages, which results in a total training time of $\approx$48 hours. We note that the first stage (I-frame) trains more than $14\times$ faster than the last stage in terms of steps/s.

Batch size	#I	#P	# steps [k]	steps/s	time [h]
8	1	0	1 000 000	19.7	14.1
8	1	1	80 000	7.3	3.0
8	1	2	220 000	3.9	15.7
8	1	3	50 000	2.6	5.3
8	1	5	50 000	1.4	9.9
Totals:			1 400 000		48.0

Table 3. Training speed/time for each stage of our model on a Google Cloud TPU. #I, #P indicates the number of I- resp. P-frames used in that stage.

[2] https://github.com/GuoLusjtu/DVC
[3] https://github.com/tensorflow/compression

A.9. Metrics

We show all metrics in a R-D style plot in Fig. 14.

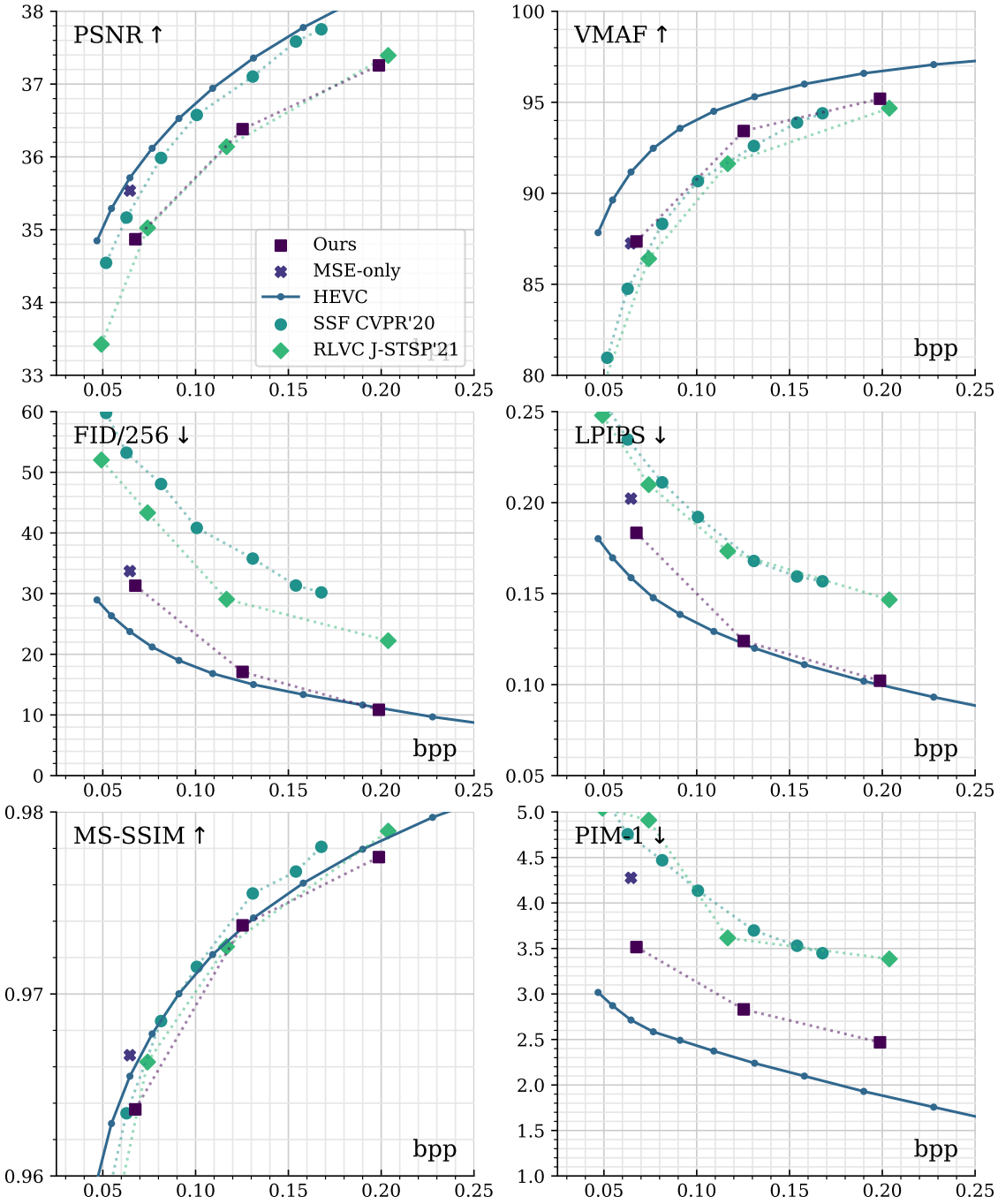


Figure 14. Metrics on MCL-JCV. We show *Ours* at three bitrates, and compare it to the *no-GAN* baseline (same as *Ours* but without GAN loss), H.264, HEVC, SSF [1], RLVC [44].

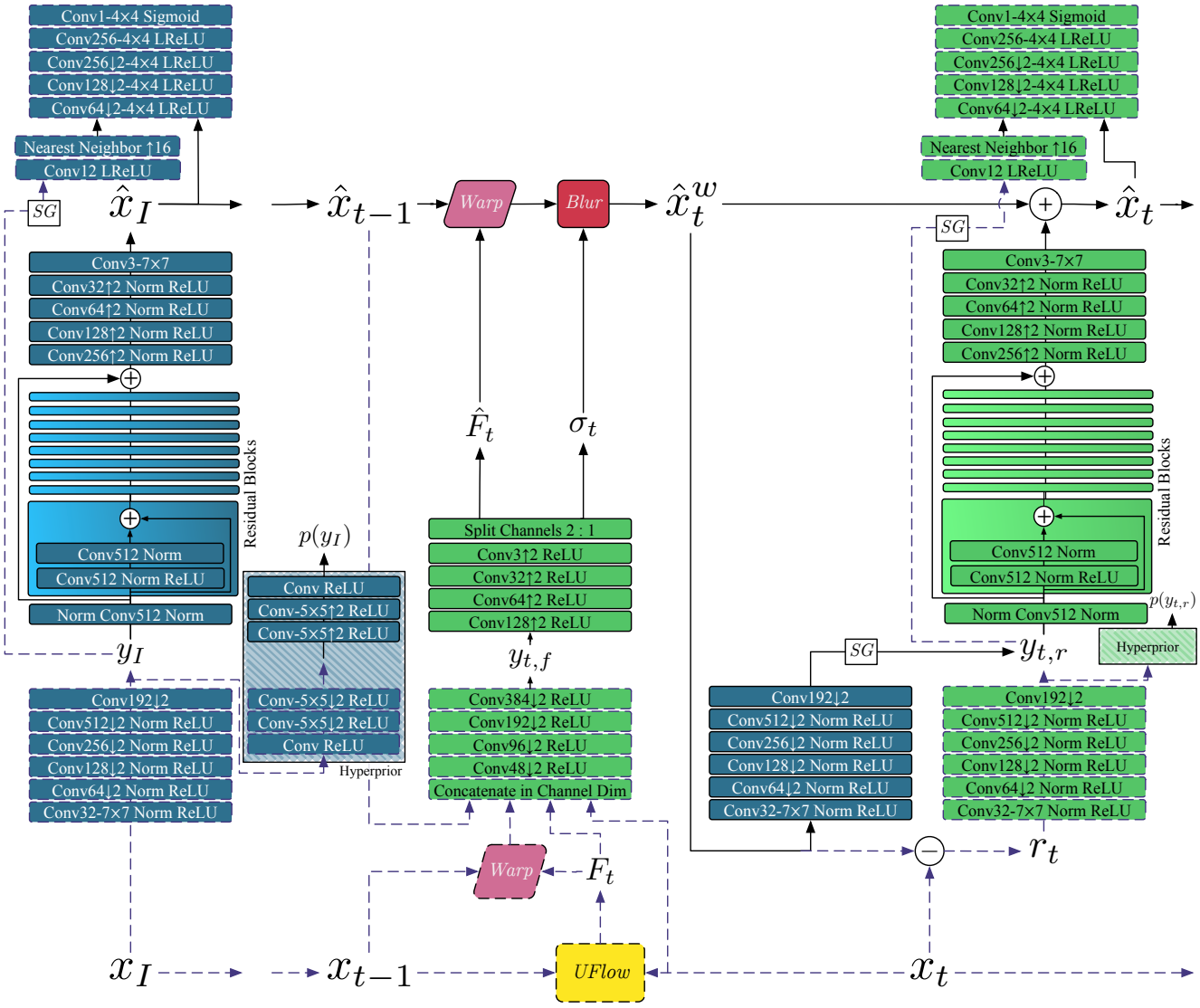


Figure 15. Detailed view of the architecture, showing the layers in each of the blocks in Fig. 2. "Conv$F$" denotes a 2D convolution with F output channels, "-S×S" denots the filter size, if that is omitted we use 3×3. ↓2, ↑2 indicates downsampling and upsampling, respectively, "Norm" is the *ChannelNorm* layer employed by HiFiC [23]. The blocks with a color gradient are Residual Blocks, we only show the detail in one. "LReLU" is the Leaky ReLU with $\alpha = 0.2$. We note that we employ SpectralNorm in both discriminators. The distributions predicted by the Hyperprior are used to encode the latents with entropy coding. Like in Fig. 2, learned I-frame CNNs are in blue, learned P-frame CNNs in green, dashed lines are not active during decoding, SG is a stop gradient operation, *Blur* is scale space blurring, *Warp* is bicubic warping. *UFlow* is a frozen optical flow model from [18].

CSVs	https://storage.googleapis.com/nvc_cvpr2022/csvs.zip
Ours	https://storage.googleapis.com/nvc_cvpr2022/ours.zip
No-GAN	https://storage.googleapis.com/nvc_cvpr2022/nogan.zip
SSF	https://storage.googleapis.com/nvc_cvpr2022/ssf.zip
H.264	https://storage.googleapis.com/nvc_cvpr2022/h264.zip
HEVC	https://storage.googleapis.com/nvc_cvpr2022/hevc.zip

Table 4. Links to user study data.

B. Data Release

B.1. CSVs and Reconstructions

For each user study comparison we made between methods, we release the reconstructions as well as a CSV containing all the rater information, via anonymous links, see Table. 4.

Reconstructions folders:
Folder per method, which contains a subfolder for each of the 30 videos of MCL-JCV, and each such video subfolder contains 60 PNGs, the reconstructions of the resp. method.

CSVs: For each study, we release a CSV, where:
Each row is a video, and we have the following columns: `wins_left`, `wins_right` indicate the number of times each method won (left is always Ours), `bpp_left`, `bpp_right`, indicate the per-video bpps, `avg_flips`, `avg_answer_time_ms`, `avg_num_pauses` indicate average flips, average time per video, and average num pauses, respectively.

B.2. Tables

We show wins per method per video, that are available in the CSVs, in Table 5.

video	Ours	No-GAN	Ours	SSF	Ours	DVC	Ours	RLVC	Ours	H264	Ours	HEVC
01	4	4	5	4	10	1	6	3	6	2	4	6
02	4	4	7	1	10	0	4	4	2	6	3	7
03	7	2	7	1	11	0	6	2	7	2	7	3
04	5	3	5	3	11	0	5	3	8	0	8	2
05	6	2	7	1	9	1	6	3	7	2	6	4
06	3	5	5	4	7	3	6	3	4	4	8	2
07	7	1	7	1	12	0	6	2	7	1	8	2
08	7	1	7	2	10	1	6	3	5	4	9	1
09	4	4	8	2	6	5	6	3	6	2	7	3
10	5	3	8	1	8	3	7	2	5	3	2	8
11	7	1	5	3	10	1	7	2	5	3	6	4
12	7	1	7	1	11	0	5	3	4	4	7	3
13	5	3	3	5	9	2	4	5	4	5	8	2
14	6	2	7	2	9	2	7	2	6	2	8	2
15	7	2	8	1	8	2	7	3	7	1	8	2
16	4	4	7	2	9	2	5	4	6	2	8	2
17	4	4	5	3	10	2	6	2	7	1	9	1
18	5	3	6	2	10	1	7	2	6	3	6	4
19	7	2	8	1	8	2	8	1	5	3	6	4
20	6	2	3	6	10	0	8	0	1	8	3	7
21	3	5	4	5	8	3	6	3	4	4	3	7
22	5	3	5	3	9	3	5	4	6	2	7	3
23	5	3	7	2	10	1	8	2	6	2	8	2
24	5	3	6	2	10	1	6	4	5	3	2	8
25	6	2	8	1	8	3	5	5	5	3	5	5
26	4	4	6	2	8	2	8	1	4	5	7	3
27	8	0	7	1	10	1	6	3	7	1	8	2
28	7	1	7	1	9	2	6	3	8	0	5	5
29	7	1	5	4	7	4	5	4	2	7	2	8
30	5	3	8	1	9	1	7	2	6	2	6	4
	165	78	188	68	276	49	184	83	161	87	184	116

Table 5. Wins per method for our user studies.