A Appendix

A.1 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.2 User Study Details

– Metrics for all ablation studies from Fig. 6 are shown in Table 2
– 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.

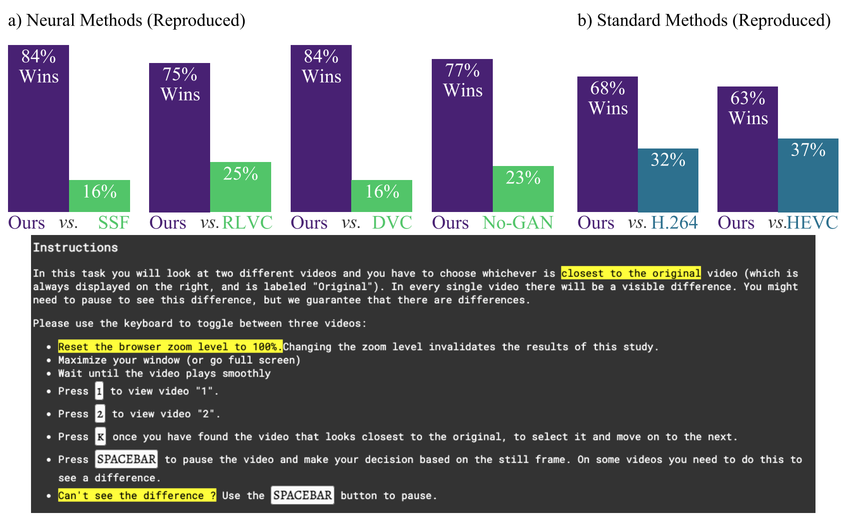

Fig. 8: *Top:* Study repeated 3 days later. *Bottom:* Instructions shown to the raters.

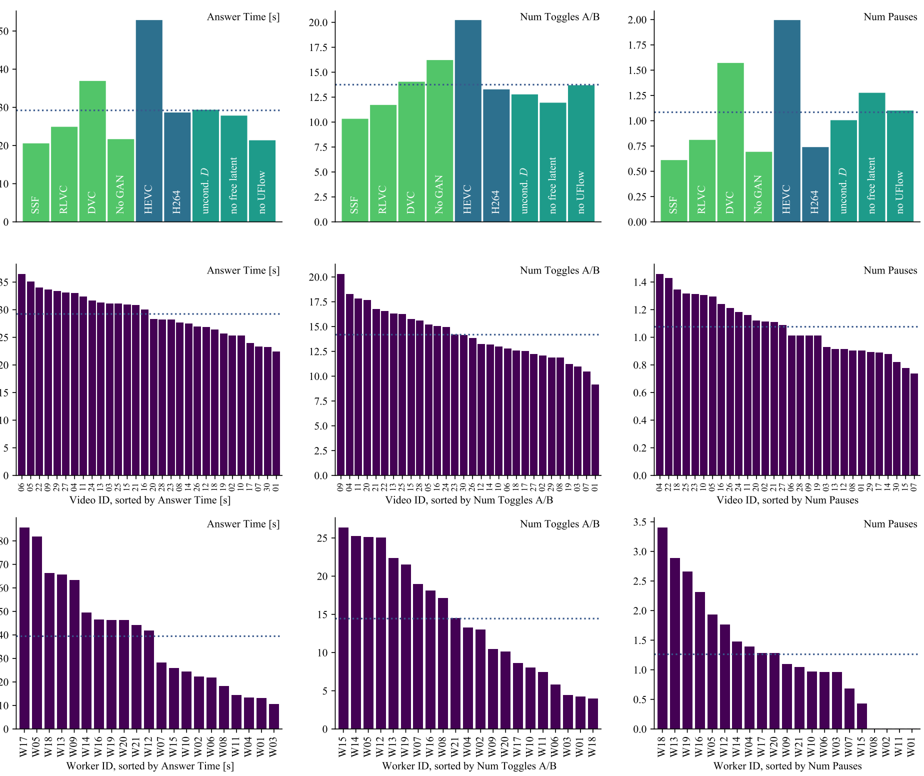

Fig. 9: User study statistics, grouped in various ways. The dotted line in each plot indicates the mean of the values shown in the plot. From left to right, we show different statistics: Answer time in [s]econds, number of flips, and number of pauses. The rows show different ways of grouping the data: *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).

	Ours	No-GAN	Predicts?	Uncond. Disc.	Predicts?	No free latent	Predicts?	No UFlow	Predicts?	HEVC	Predicts?	HEVC	Predicts?
PSNR↑	34.5	35.1	*No*	34.8	*No*$_{\approx}$	33.9	*Yes*	33.9	*Yes*	37	*No*	38.2	*No*
MS-SSIM↑	0.964	0.967	*No*$_{\approx}$	0.966	*No*$_{\approx}$	0.959	*No*$_{\approx}$	0.96	*No*$_{\approx}$	0.974	*No*$_{\approx}$	0.979	*No*$_{\approx}$
VMAF↑	87.3	86.9	*No*$_{\approx}$	85.6	*Yes*	81.9	*Yes*	84.2	*Yes*	94.7	*No*	96.5	*No*
PIM-1↓	3.34	4.17	*Yes*	3.83	*Yes*	3.85	*Yes*	3.32	*No*$_{\approx}$	2.15	*No*	1.75	*No*
LPIPS↓	0.168	0.194	*Yes*	0.172	*Yes*	0.194	*Yes*	0.167	*No*$_{\approx}$	0.112	*No*	0.0895	*No*
FID/256↓	32.8	35.7	*Yes*	34.9	*Yes*	35.9	*Yes*	32.7	*No*$_{\approx}$	15.5	*No*	10.7	*No*
Preferred vs. Ours↑		32%		28%		23%		33%		41.2%		41.4%	

Table 2: We show metrics corresponding to the user studies, where the last row repeats the results from Fig. 6. We indicate whether each metric *predicts* the study, using *Yes* and *No*. 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. We can see that no metric predicts all user studies (since *Ours* is preferred in all studies).

A.3 Decoupled Scale-Space Warping: Details

```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)
```

Fig. 10: Numpy implementation of adaptive blurring.

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.

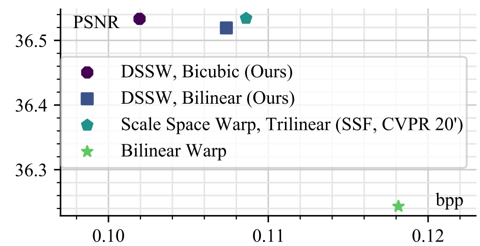

Fig. 11: To validate our Decoupled Scale Space Warping (DSSW) implementation, we train models **for MSE**. We compare the R-D performance of Bilinear/Bicubic resamplers in DSSW against the Scale-Space Warping of Agustsson *et al*. [1] and find that DSSW with bicubic improves the bpp. We also show plain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

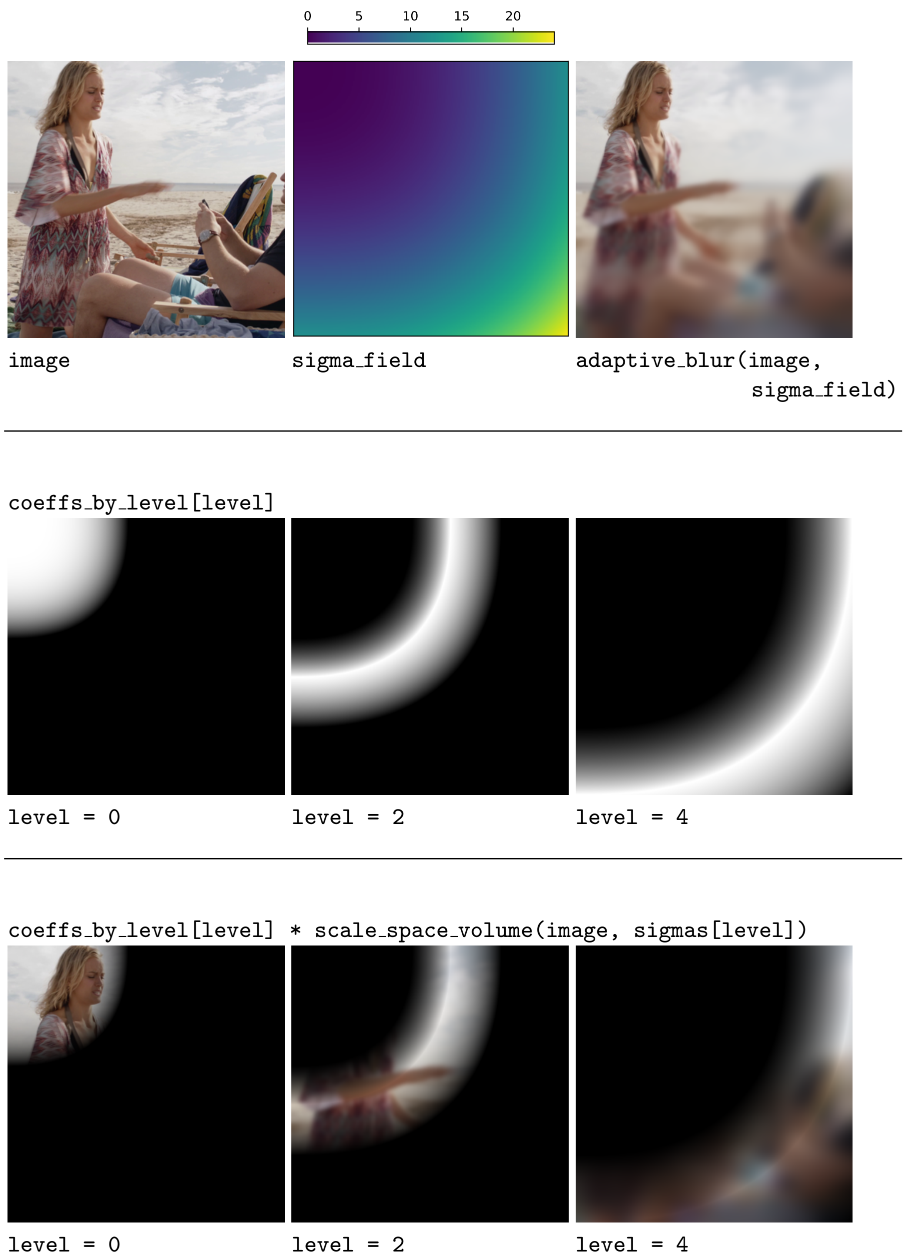

Fig. 12: Visualizing variables of the algorithm given in Fig. 10.

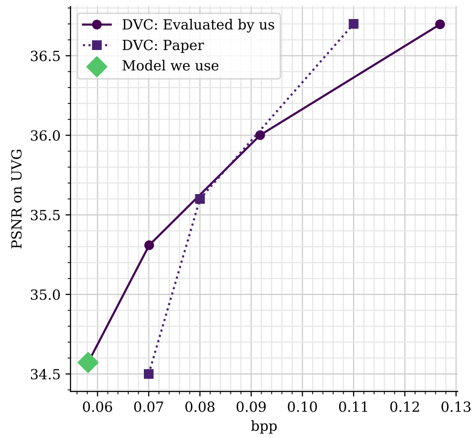

Fig. 13: Comparing our DVC models to what the authors reported, on UVG. We use the model in the lower right, as this is closest to our bpps (achieving ≈ 0.06bpp on UVG, ≈ 0.09bpp on MCL-JCV).

A.4 DVC Details

To get DVC reconstructions, we use the code provided by the authors.[1] 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.[2]. 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.5 Architecture Details

A detailed version of the architecture from Fig. 3 is given in Fig. 14.

A.6 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 [27,24] 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

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

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 $k_{\mathrm{TV}} = 10.0, k_{\mathrm{flow}} = 1.0$ in $\mathcal{L}_{\mathrm{reg}}$. 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 = 1E−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 = 1E−5 after $320k$ steps. We train the discriminators for 1 step for each generator training step.

A.7 Training Time

In Table 3 we report the training speed for each of the training stages, which results in a total training time of ≈48 hours. We note that the first stage (I-frame) trains more than 14× 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.

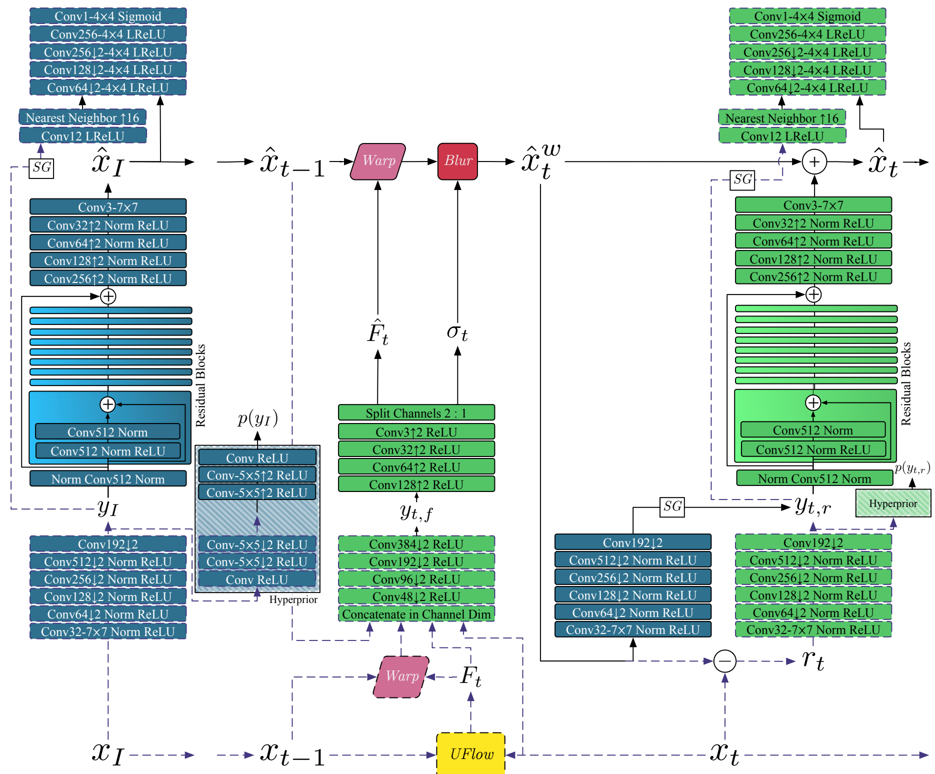

Fig. 14: Detailed view of the architecture, showing the layers in each of the blocks in Fig. 3. "Conv$F$" denotes a 2D convolution with F output channels, "-S×S" denots the filter size, if that is omitted we use 3×3. ↓2, ↑2 indicates downsampling and upsampling, respectively, "Norm" is the *ChannelNorm* layer employed by HiFiC [24]. 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. 3, 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].

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.

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

Table 4: Links to user study data.

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.