C3: High-performance and low-complexity neural compression from a single image or video

Supplementary Material

Acknowledgements

We would like to thank: Wei Jiang for providing RD and MACs/pixels numbers for several baselines both on the Kodak and CLIC2020 datasets; Fabian Mentzer and Ho Man Kwan for providing video RD numbers for various baselines; Eirikur Agustsson for helping to run VTM (RA); COOL-CHIC authors for open sourcing their code; Yee Whye Teh, Nick Johnston, Fabian Mentzer and Eirikur Agustsson for helpful feedback.

A. Method and experimental details

A.1. Model architecture

Here we provide full details of all model components of C3, cf. Fig. 2 in the main paper for a visualization of the model. In App. A.4 we provide all hyperparameter settings for our experiments.

A.1.1 Multi-resolution latent grids

We follow Cool-chic and structure the latent $\mathbf{z}$ in a hierarchy of N latent grids, $\mathbf{z}^1, \dots, \mathbf{z}^N$, at multiple resolutions. Each latent grid $\mathbf{z}^n$ has a single channel and is of shape (h_n, w_n) for images and (t_n, h_n, w_n) for videos. By default, these sizes are related to the image shape (h, w) or video shape (t, h, w) through

$$(t_n, h_n, w_n) = (\tfrac{t}{2^{n-1}}, \tfrac{h}{2^{n-1}}, \tfrac{w}{2^{n-1}}), \qquad n = 1, \dots, N, \tag{6}$$

that is, the latent grid $\mathbf{z}^n$ is a factor 2 smaller in each dimension than the previous grid $\mathbf{z}^{n-1}$. All latent grids are initialized to zero at the start of optimization.

A.1.2 Upsampling the latent grids to the resolution of the input

Each latent grid is deterministically upsampled to the input resolution $(\{t\}, h, w)$ before all grids are concatenated together and passed as input to the synthesis network. Cool-chic uses simple bicubic upsampling for this [43]. Cool-chic v2 instead uses learned upsampling that is implemented as a strided convolution (allowing for upsampling by a factor of 2 only) that is initialized to bicubic upsampling [48].

We experimented with different forms of upsampling (both learned and fixed) in our setup but found that for almost all bitrates, using simple bilinear upsampling led to the best results. More complex upsampling methods such as bicubic or Lanczos upsampling only led to better results for very low bitrates. We explain this observation as follows. For high bitrates, fine details are modeled by the highest resolution latent grid, which already matches the resolution of the input, see, *e.g.*, the top row in Fig. 26. Therefore bilinear upsampling of the lower resolution latents is sufficient. For low bitrates, only very few details are modeled by the highest resolution latent grid, see bottom row in Fig. 26. The model therefore relies much more heavily on information upsampled from the lower resolution latent grids to explain fine details; in this case, more complex upsampling methods are beneficial. Because the differences even for lowest bitrates were very small, we opted to exclusively use bilinear interpolation as it also has a lower decoding complexity. For videos we use trilinear interpolation.

A.1.3 Image synthesis with the synthesis network

The upsampled latents are stacked into a single tensor of shape $(\{t\}, h, w, N)$ and are then used as input for the synthesis network f_θ, which directly predicts the raw RGB intensity values (output values are clipped to lie in the correct range).

To parameterize f_θ, Cool-chic uses a simple MLP that is applied separately to each of the $(\{t\}, h, w)$ pixel locations. This operation can be equivalently implemented as a sequence of 1×1 convolutions. In addition, Cool-chic v2 optionally adds several 3×3 residual convolutions. For the residual convolutions the input and output channel dimensionality is set to 3 to keep the decoding complexity low. C3 follows the same architecture layout but uses the more expressive GELU [33] activation

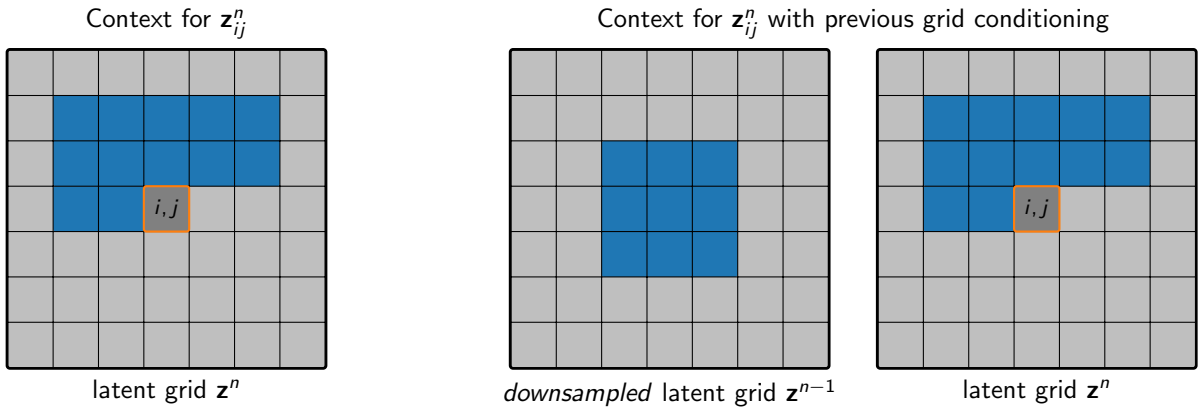

Figure 10. Illustration of the context used by the entropy model to predict the distribution parameters of a latent grid location $\mathbf{z}_{ij}^n$. *Left:* Without previous grid conditioning the context only comes from a (causal) neighborhood in the same latent grid. *Right:* With previous grid conditioning a small neighborhood from the bilinearly downsampled latent grid $\mathbf{z}^{n-1}$ is used in addition to the context from the current latent grid $\mathbf{z}^n$.

function instead of ReLUs. We also opt to use narrower and deeper networks with a similar overall decoding complexity. For C3, the 1×1 convolutions are initialized with the standard He initialization while the residual convolutions are initialized to zero.

A.1.4 Entropy model for the latent grids

The entropy model is used to losslessly compress the (quantized) latent grids into a bitstream. Each latent grid location, $\mathbf{z}_{ij}^n$ for images and $\mathbf{z}_{\tau ij}^n$ for videos, respectively, is entropy en-/decoded using a quantized Laplace distribution with mean parameter μ_{ij}^n ($\mu_{\tau ij}^n$ for video) and scale parameter σ_{ij}^n ($\sigma_{\tau ij}^n$ for video). Both parameters are autoregressively predicted from the context of the latent grid entry using the entropy network g_ψ:

$$\mu_{ij}^n, \sigma_{ij}^n = g_\psi\left(\text{context}\left(\mathbf{z}^n, (i,j)\right)\right). \qquad \text{for images} \tag{7}$$

$$\mu_{\tau ij}^n, \sigma_{\tau ij}^n = g_\psi\left(\text{context}\left(\mathbf{z}^n, (\tau,i,j)\right)\right). \qquad \text{for videos} \tag{8}$$

Because the autoregressive prediction also occurs at decoding time, the context has to be causally masked/extracted; following Cool-chic, we use a (causally masked) neighborhood of the latent entry as its context (cf. Fig. 10 *(left)*). g_ψ is a fully connected network that maps the context to the Laplace distribution parameters. During optimization and encoding, when all latent entries are available, we use 1×1 convolutions ($1 \times 1 \times 1$ for video) with the corresponding number of channels to replace the MLP and use a masked $k_h \times k_w$ convolution ($k_t \times k_h \times k_w$ for video) to replace the extraction of context of size k and the first layer of the MLP, cf. Fig. 10 *(left)*.

The scale parameter σ of the Laplace distribution is constrained to be positive while the output of the entropy network g_ψ in unconstrained and can be positive and negative. We therefore pass the raw prediction of the network through an exponential function; in other words, the network predicts the log-scale instead of the scale as is usually done in practice when parameterizing positive values. We found that shifting the predicted raw log-scale value by a constant can improve optimization dynamics as it determines the behavior of the model close to initialization.

So far the context for an entry $\mathbf{z}_{ij}^n$ only takes into account the local neighborhood in the same grid $\mathbf{z}^n$. This means that the entropy model factorizes across grids:

$$P_\psi(\mathbf{z}) = \prod_n P_\psi(\mathbf{z}^n). \tag{9}$$

Alternatively, we also consider the option of extending this context to include local grid entries in a neighboring grid $\mathbf{z}^{n-1}$:

$$P_\psi(\mathbf{z}) = P_\psi(\mathbf{z}^1) \prod_{n>1} P_\psi(\mathbf{z}^n | \mathbf{z}^{n-1}). \tag{10}$$

Note that because of the autoregressive structure, we can only depend on the neighboring grid in one direction. In that case, we downsample the previous latent grid $\mathbf{z}^{n-1}$ to the same resolution as the current grid, $\mathbf{z}^n$, and extract the full neighborhood around the location of interest from it; cf. Fig. 10 *(right)* for an illustration. The size of the neighborhood extracted from the previous grid can vary from the size of the neighborhood in the current grid. We found that in practice, small neighborhood sizes for the previous grid (*e.g.* 3×3) were sufficient.

We also explored the option of using separate entropy parameters for different grids

$$P_\psi(\mathbf{z}) = \prod_n P_{\psi^n}(\mathbf{z}^n), \tag{11}$$

and found that while this did not help for images, it gave better RD performance on videos, especially when using different masking patterns for different grids (cf. App. A.3).

A.2. Quantization-aware optimization of the rate-distortion objective

Here, we provide further details about the RD objective in Eq. (4) that is used to fit the latent grids $\mathbf{z}$ as well as the synthesis network f_θ and the entropy network g_ψ to a particular image $\mathbf{x}$ using gradient-based optimization. We reproduce the objective here for easier reference:

$$\mathcal{L}_{\theta,\psi}(\mathbf{z}) = \|\mathbf{x} - f_\theta\left(\text{Upsample}(\mathbf{z})\right)\|_2^2 - \lambda \log_2 P_\psi(\mathbf{z}). \tag{4}$$

The objective trades off reconstruction of the image (first term) and compression of the latent $\mathbf{z}$ (second term) with an RD-weight λ. By varying λ we trace out different points in the RD-plane that we plot as RD-curves. High values of λ lead to low bitrates and vice versa. Note that the objective does not take into account the quantization of the model parameters θ and ψ.

At evaluation time, the distortion (measured in PSNR) and the rate are computed from quantized versions of the variables, $\widehat{\mathbf{z}}, \widehat{\theta}$, and $\widehat{\psi}$, that have been entropy-decoded from the bitstream, see Fig. 2 for an illustration of the decoding.

During optimization we evaluate the RD-loss in Eq. (4) using continuous variables $\mathbf{z}, \theta, \psi$. Naive minimization of the objective w.r.t. the continuous variables would give rise to a solution that does not work well when the variables are quantized. Instead, we have to take the subsequent quantization into account during optimization. In practice, quantization of the latent grid values $\mathbf{z}$ is most relevant, such that the optimization is only made aware of the quantization of the latent grids and not of the quantization of the network parameters themselves. We explain how these network parameters are quantized and entropy en-/decoded in App. A.2.6.

A.2.1 Quantized Laplace distribution for continuous variables: Integrated Laplace distribution

As explained above, the rate is modeled by (the log density of) a quantized Laplace distribution P_ψ whose distribution parameters are predicted by the entropy model g_ψ.

During optimization we use continuous values and, following Ballé et al. [6] and Ladune et al. [43], replace the quantized Laplace distribution with an integrated Laplace distribution that integrates the probability mass over the rounding/quantization interval. When evaluated on quantized values, the two distributions are identical:

$$P_\psi(\mathbf{z}_{ij}^n) = \int_{\mathbf{z}_{ij}^n - 0.5}^{\mathbf{z}_{ij}^n + 0.5} \text{Laplace}(z; \mu_{ij}^n, \sigma_{ij}^n)\, \mathrm{d}z, \tag{12}$$

where the location parameter μ_{ij}^n and the scale parameter σ_{ij}^n of the Laplace distribution are autoregressively predicted by the entropy network g_ψ.

A.2.2 Two stages of optimization

Following Cool-chic and Cool-chic v2, we split the optimization into two stages that differ in how the quantization of the latents is approximated. As discussed in the main paper, we make improvements to both stages. Here, we explain the two stages used by C3 in details; we also highlight the main differences to prior work where appropriate.

Stage 1: Soft-rounding $\mathbf{z}$ and adding noise to it. In the first stage, the continuous values for the latent grids $\mathbf{z}$ are passed through an invertible soft-rounding function and additionally perturbed with additive noise as we describe in Apps. A.2.4 and A.2.5, respectively. Because the soft-rounding function is invertible and differentiable everywhere, we can compute its gradient with backpropagation, and the additive noise can be ignored for the gradient computation as it does not depend on any of the parameters (the soft-rounding function is implemented in a reparameterized form):

$$\text{forward}: \quad \mathcal{L}_{\theta,\psi}(\text{softround}_T(\mathbf{z}, \mathbf{u})) \qquad \mathbf{u} \sim p_{\text{noise}}(\mathbf{u}) \tag{13}$$

$$\text{backward for } \theta, \psi: \quad \nabla_{\theta,\psi}\, \mathcal{L}_{\theta,\psi}(\text{softround}_T(\mathbf{z}, \mathbf{u})) \tag{14}$$

$$\text{backward for } \mathbf{z}: \quad \nabla_{\mathbf{z}}\, \mathcal{L}_{\theta,\psi}(\text{softround}_T(\mathbf{z}, \mathbf{u})) \tag{15}$$

Gradient variance is a concern in this stage of training, especially since we use larger learning rates. Because of this, we cannot use a temperature T in the soft-rounding that is too low (cf. App. A.2.4 for details), and we also found it beneficial to use more concentrated noise distributions than uniform noise early in training (cf. App. A.2.5 for details).

Cool-chic and Cool-chic v2 do not use soft-rounding and instead directly add uniform noise, $p_{\text{noise}}(\mathbf{u}) = \text{Uniform}(\mathbf{u})$.

Stage 2: Hard-rounding $\mathbf{z}$. In the second stage, the continuous values for the latent grids $\mathbf{z}$ are (hard-)rounded; *i.e.*, they are replaced by their quantized values $\widehat{\mathbf{z}} = \lfloor \mathbf{z} \rceil$. Quantizing the latents increases the variance of the gradients w.r.t. the network parameters θ and ψ and necessitates lower learning rates as we discuss in App. A.2.3. To estimate gradients w.r.t. the latent $\mathbf{z}$, we have to backpropagate through the discrete rounding; as this is not possible, we replace the hard-rounding with soft-rounding using a very low temperature $T = 10^{-4}$ for this case. This estimator approximates the hard-rounding well but is invertible; however, it is still biased. Note that we do not add any noise when using this soft-rounding estimator.

$$\text{forward}: \quad \mathcal{L}_{\theta,\psi}(\lfloor \mathbf{z} \rceil) \tag{16}$$

$$\text{backward for } \theta, \psi: \quad \nabla_{\theta,\psi}\, \mathcal{L}_{\theta,\psi}(\lfloor \mathbf{z} \rceil) \tag{17}$$

$$\text{backward for } \mathbf{z}: \quad \nabla_{\mathbf{z}}\, \mathcal{L}_{\theta,\psi}(\text{softround}_{T\to 0}(\mathbf{z})) \tag{18}$$

Because of our improvements to stage 1, the loss as well as the corresponding rate and distortion values do not change by much in the second stage of optimization. Overall, stage 2 seems to be less important for C3 than for Cool-chic v2, though it still leads to small improvements. See App. D.2 for an ablation.

Cool-chic also uses the quantized latent $\widehat{\mathbf{z}}$ in the second stage but uses simple (linear) straight-through estimation to estimate gradients w.r.t. $\mathbf{z}$ [43]; the linear function is a cruder approximation of (hard) rounding than the soft-rounding function, such that the bias of this estimator is larger than for C3. Cool-chic v2 also uses linear straight-through estimation but downscales the gradient by a factor $\epsilon \ll 1$ [48]. This results in the same biased straight-through estimator but effectively changes the learning rate of the latents to be smaller.

A.2.3 Learning rate decay

As for most optimization based algorithms the learning rate is one of the most important hyperparameters in C3. We use two simple strategies to choose the learning rate for stage 1 and stage 2, respectively, cf. Fig. 11 for a schematic.

Learning rate in Stage 1. We use a cosine decay schedule that starts at a higher value and is decayed to 0 throughout stage 1, which also makes up most of the optimization steps. The initial learning rate value is chosen empirically.

Learning rate in Stage 2. Due to the variance of the gradients and the bias of the estimator, the second stage of optimization depends even more strongly on the learning rate. We found that starting with a high enough learning rate was important to make progress, but that an aggressive decay of the learning rate may be necessary as otherwise the loss can quickly get worse. Instead of using a fixed schedule, we therefore opted for the following automatic and adaptive mechanism: Starting from a fixed sufficiently high learning rate (10^{-4} in our experiments), we track the loss and decay the learning rate by a fixed factor if the loss does not improve for a certain number of steps. Upon decaying the learning rate we also reset the parameters $(\mathbf{z}, \theta, \psi)$ and optimizer state to their previous best values as measured by the loss. The stage finishes after a certain number of steps or when the learning rate is decayed below a certain threshold value.

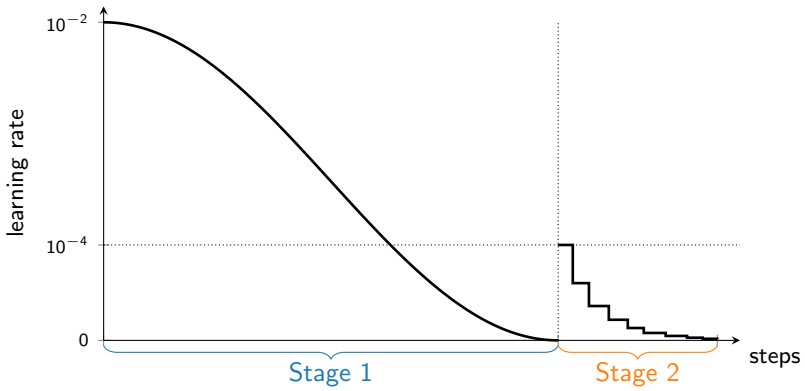

Figure 11. Schematic of the learning rates used for the two stages of optimization. Axes are not to scale and values are only indicative. In stage 1, the learning rate is decayed with a cosine schedule from an initial value to 0 at the end of stage 1. In stage 2 we adaptively decay the learning rate by a constant factor whenever the evaluation loss does not improve for a certain number of steps.

A.2.4 Soft-rounding

As discussed above, we warp the continuous latent values $\mathbf{z}$ during the first stage of optimization with a soft-rounding function to better approximate the eventual quantization of the latents. A soft-rounding function is a differentiable relaxation of the (hard) rounding function; that is, it has a parameter T (typically referred to as *temperature*) whose value determines how well we approximate the hard rounding function. Crucially, by setting T to a particular value ($T = 0$ in our case), we recover the hard rounding function. As $T \to \infty$ our soft-rounding function (see below) tends to a linear function, equivalent to the straight-through gradient estimator that is used in Cool-chic [43]. Note that despite using a soft-rounding function, we still have to add random noise to regularize the optimization. We explain this further in App. A.2.5.

Following Agustsson and Theis [2], we use a construction where we apply soft-rounding twice: first to the raw value $\mathbf{z}$ and a second time after adding random noise $\mathbf{u}$ to the result. That is, the $\mathrm{softround}_T(\mathbf{z}, \mathbf{u})$ function in Eqs. (13) to (15) is given by

$$\mathrm{softround}_T(\mathbf{z}, \mathbf{u}) = r_T(s_T(\mathbf{z}) + \mathbf{u}), \tag{19}$$

where r_T and s_T are simple soft-rounding functions.

Following Agustsson and Theis [2], we used the following simple soft-rounding function,

$$s_T(z) = \lfloor z \rceil + \frac{1}{2}\frac{\tanh(\Delta/T)}{\tanh(1/2T)} + \frac{1}{2}, \quad \Delta = z - \lfloor z \rfloor - \frac{1}{2}, \tag{20}$$

which is invertible and differentiable everywhere. We further also use

$$r_T(y) = s_T^{-1}(y - 0.5) + 0.5 \approx \mathbb{E}_X[X \mid s_T(X) + U = y] \tag{21}$$

for the second soft-rounding function (instead of applying s_T again) as suggested by Agustsson and Theis [2]. Here, X and U are assumed to be uniform random variables. We found that $s_T(s_T(z) + u)$ seemed to perform equally well in our setting.

As the learning rate is decayed throughout stage 1, we can also decrease the temperature T of the soft-rounding to better approximate the rounding operation. For simplicity we use a linear schedule that interpolates between a higher temperature ($T = 0.3$) at the beginning of stage 1 and a lower temperature ($T = 0.1$) at the end of stage 1. A higher temperature corresponds to a more linear function while a lower temperature leads to a more step-like function, cf. Fig. 3.

A.2.5 Kumaraswamy noise distribution

As discussed in Sec. 3.1, adding noise during stage 1 is necessary even with soft-rounding because the (invertible) soft-rounding function alone does not create an information bottleneck. What do we mean by this? Hard rounding irreversibly destroys information by mapping all latent values in a certain quantization bin to the same (quantized) value. Downstream computations, such as the reconstruction of the image with the synthesis network, then only have access to these quantized values. Therefore, it is important that the synthesis and entropy network are optimized in such a way as to only rely on the information in

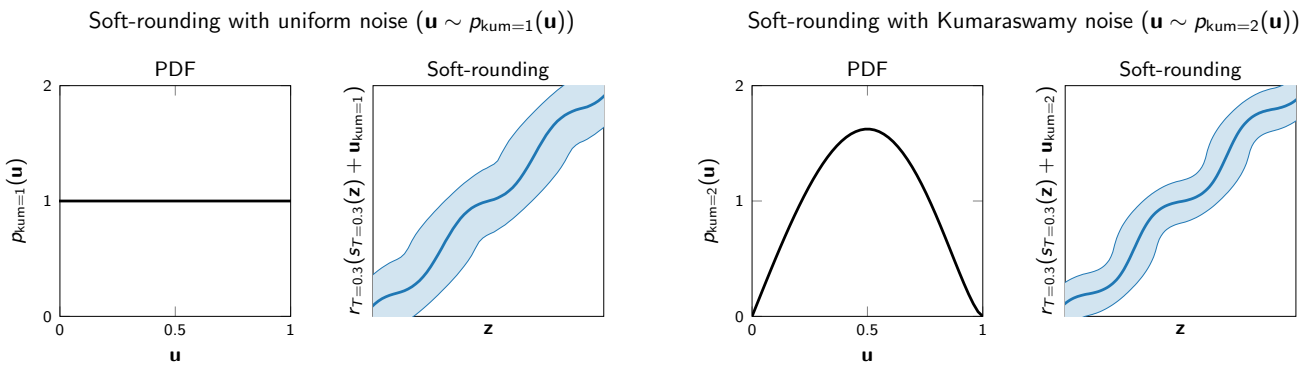

Figure 12. Probability density function (PDF) of the simplified Kumaraswamy distribution, cf. Eq. (23), and the effect of using it in the soft-rounding function (Eq. (19)) for two values of the shape parameter a. *left:* $a = 1$ corresponds to the uniform distribution; *right:* $a = 2$ yields a more peaked distribution that results in reduced variability of the soft-rounding function. For the soft-rounding we plot the mean and its 95-percentiles.

the quantized values, rather than information about the precise location of the latent within the quantization bin. During optimization we use continuous valued latents as well as a continuous and invertible relaxation of rounding (as discussed in App. A.2.4). And while the soft-rounding function can be steep for low temperatures T, its warping can be undone; that is, the synthesis network could learn to invert the soft-rounding function and then rely on information about location within the bin to improve the distortion loss without sacrificing the rate loss. The addition of noise is a mechanism to destroy this information in a *differentiable* manner, such that we can still evaluate gradients of the objective, but prevent the networks from learning to use this information (that will get destroyed with quantization).

A consequence of adding noise is that the gradients of the objective (Eqs. (14) and (15)) become stochastic, and the strength of the noise determines the variance of these gradients. Large gradient variance can lead to slower or worse optimization. In particular when using the soft-rounding function, there is no reason *a priori* that uniform noise should strike the best balance. We therefore explored other noise distributions as the detail in the following.

We want to flexibly parameterize the noise distribution in the compact interval $[0, 1]$. A natural choice for this is the $\text{Beta}(a, b)$-distribution that has two shape parameters a and b and can represent the Uniform distribution as well as symmetric and asymmetric overdispearsed (spread out) and underdispersed (peaked) distributions. However, we found that sampling from the Beta-distribution is slow due to the transcendental functions involved in computing its density and CDF. We therefore use the Kumaraswamy distribution [41] that is similar to the Beta-distribution but has a closed form density and CDF.

The Kumaraswamy probability density function also has two shape parameters, a and b, and is given by

$$p_{a,b}(u) = abu^{a-1}(1 - u^a)^{b-1}. \tag{22}$$

We are only interested in distributions with a mode at 0.5 so as not to favor one direction; we can therefore simplify the distribution to only have a single parameter a:

$$p_{\text{kum}=a}(u) = (2^a(a-1)+1)u^{a-1}(1-u^a)^{\frac{1}{a}(2^a-1)(a-1)}. \tag{23}$$

Setting $a = 1$ corresponds to the uniform distribution, $p_{\text{kum}=1}(u) = \text{Uniform}(u)$. We plot this simplified Kumaraswamy distribution (Eq. (23)) for $a = 1$ and $a = 2$ in Fig. 12. Note that while the mode is at $u = 0.5$, the distribution is not quite symmetric; yet we observed that this did not matter in practice, likely because the asymmetry is small.

In Fig. 12 we also show the effect of sampling from these distributions on the soft-rounding; as expected, sampling from a more peaked distribution, $p_{\text{kum}=2}$, leads to smaller uncertainty intervals at the same temperature ($T = 0.3$ in this case) for the soft-rounding.

Because gradient variance is of more concern at high learning rates at the beginning of stage 1, the trade-off between regularization and optimization dynamics changes throughout stage 1. Empirically we found that linearly decaying the shape parameter a from $a = 2$ at the beginning of stage 1 to $a = 1$ (uniform distribution) at the end of stage 1 performed best.

A.2.6 Quantization and entropy encoding/decoding of the network parameters

Following Cool-chic, we treat the synthesis and entropy parameters θ, ψ as continuous values during training, and quantize them separately after training. We do a grid search over the quantization steps for the weights and bias terms for θ and ψ (so two terms in total, one for weight terms in either θ or ψ and one for bias terms in either θ or ψ), that give quantized parameters $\widehat{\theta}$ and $\widehat{\psi}$. We select the quantization step that minimizes the following modified objective:

$$\mathcal{L}'_{\widehat{\theta},\widehat{\psi}}(\mathbf{z}) = \|\mathbf{x} - f_{\widehat{\theta}}(\text{Upsample}(\mathbf{z}))\|_2^2 - \lambda(\log_2 P_{\widehat{\psi}}(\mathbf{z}) + \log P(\widehat{\theta}) + \log P(\widehat{\psi})) \tag{24}$$

Note that the RD-objective and the optimization thereof are not quantization-aware with respect to these network parameters. Addressing this may constitute interesting future work.

A.3. Video: learning the custom masking

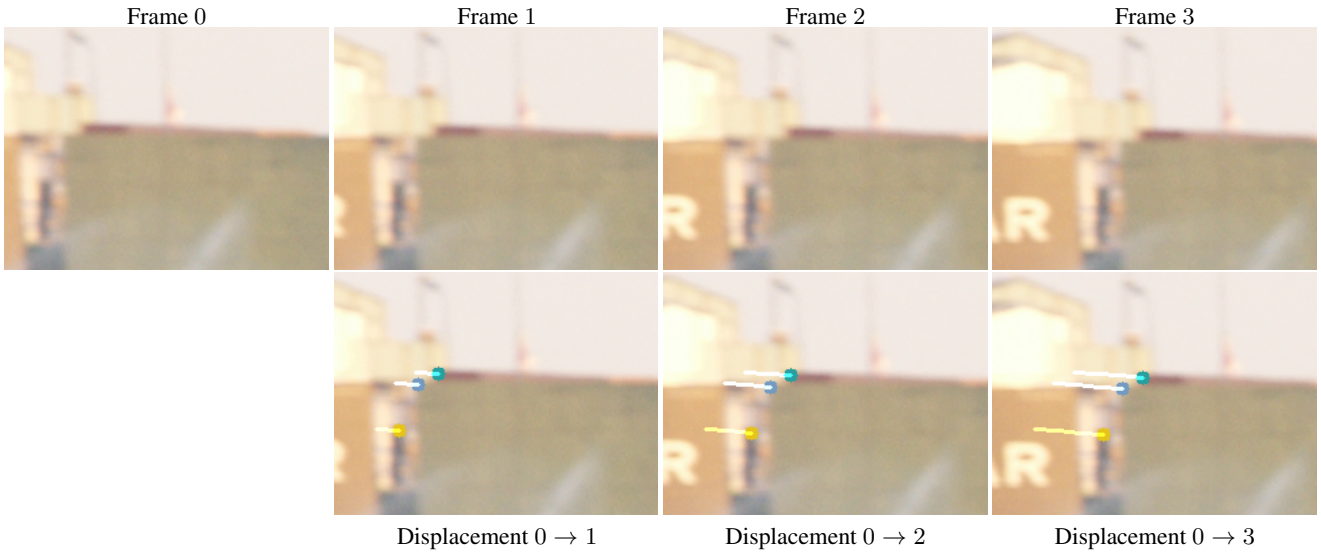

Figure 13. (Top) First few frames of a video patch from the Jockey sequence (UVG). (Bottom) displacement of key-points between consecutive frames computed using the OpenCV [36] implementation of Lucas-Kanade optical flow estimation [56].

In the video setting, the context for predicting latent entries for a particular frame can also contain entries from the previous frame (cf. Fig. 4). As discussed in Sec. 3.3, it is important that the context is wide enough for the context of the previous frame to contain relevant information for predicting the latent entry of a particular frame. For example, consider the first few frames of the video patch shown in the first row Fig. 13. The second row shows the displacement of key-points between consecutive frames, and we can see that the displacement is greater than the small context width ($5 - 7$ latent pixels) that we use for images. In fact, the displacement of key-points between consecutive frames in the second row can be quantified using the Lucas-Kanade method for optical flow estimation [56]. This gives a mean displacement of $(19.5, 0.6)$ pixels per frame in the x and y direction respectively, where the mean is taken across the first 30 frames. Given that the latents and the synthesis network are designed in such a way that the latents only contain very local information about the video pixels, we would not be able to use the previous latent frame for predicting the current latent frame with a small context width. Hence we use a larger context width to be able to better capture motion.

The issue with naïvely using a larger context is that the number of entropy parameters grows with the context size, as we need c_{hidden} entropy parameters for every context entry. Given that most of the latent entries in this wide context are irrelevant for predicting the target latent entry, we learn a custom mask for the previous latent frame context such that the entropy model can still access the relevant context in the previous latent frame while ignoring the irrelevant context therein.

Here we describe the procedure for learning this custom mask, with an overview in Fig. 14:

1. **Train C3 with wide spatial context for a few iterations.** First, train the entropy model with causal masking using a wide spatial context of size $C \times C$ per latent grid for M iterations, where M is small. Typically we use $C = 65$. Note that the wide context applies to both the previous latent frame $\tau - 1$ and the current latent frame τ. We train with separate entropy parameters for each latent grid.

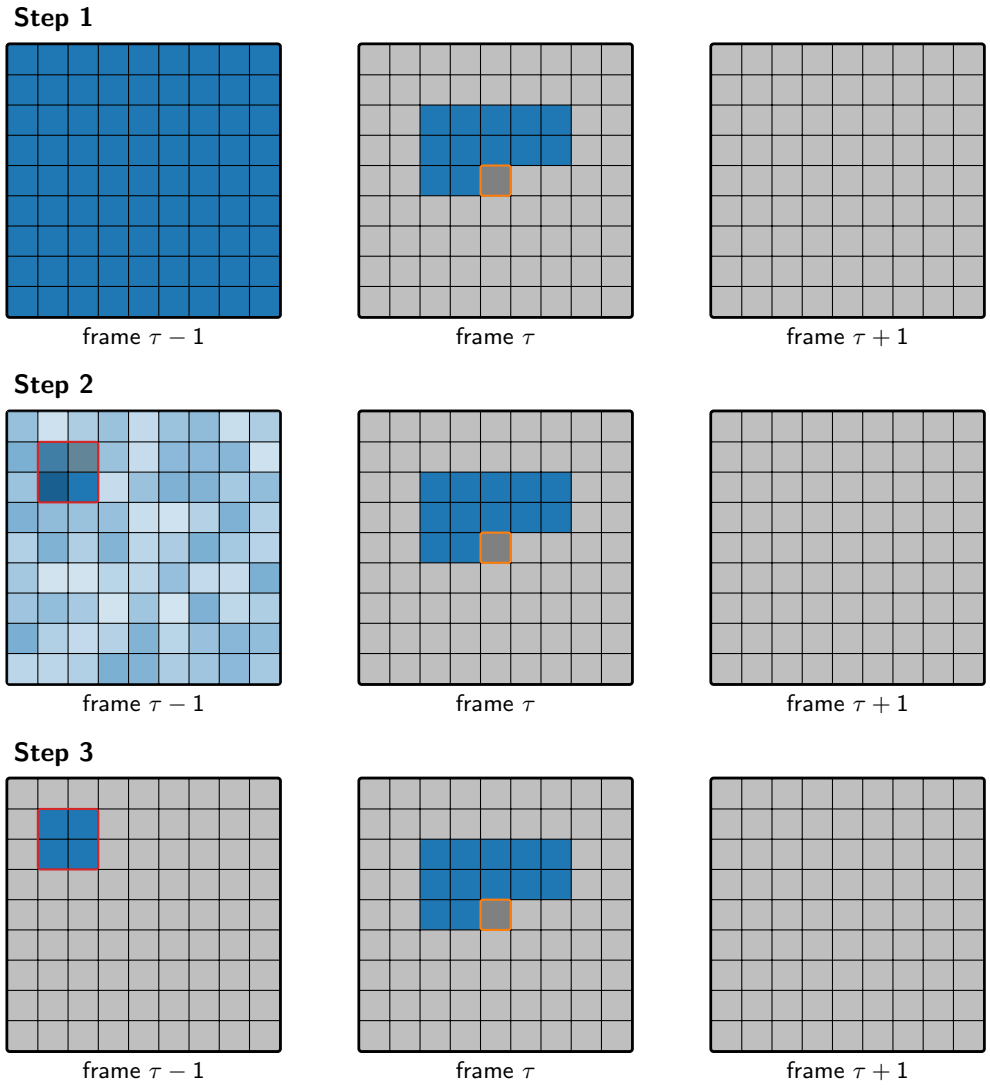

Figure 14. Visualization of the 3-step procedure for learning the custom masking.

2. **Compute magnitudes of entropy model weights for each context dimension.** For each latent grid, take the C^2 dimensions of the previous latent frame $\tau - 1$, and for each dimension compute the mean magnitude of the entropy model's first layer weights that process this dimension. *i.e.*, suppose the entropy model's first 1×1 Conv layer for the previous latent frame context has weights of shape $C \times C \times 1 \times f_{\text{out}}$. Take the absolute value of these weights, followed by the mean across the final two axes to obtain the $C \times C$ magnitudes for each of the C^2 context dimensions.
3. **Choose rectangular mask location that maximizes the sum of magnitudes within mask.** Given a fixed rectangular mask shape $c \times c'$ (where $c, c' \ll C$), sweep over all possible locations of the rectangular mask within the $C \times C$ context grid. For each location, compute the sum of the $c \times c'$ magnitudes within the mask. Then choose the location that has the highest sum.

We thus obtain the $c \times c'$ learned rectangular mask for the previous latent frame $\tau - 1$. We also empirically observed that for the lower resolution latent grids, there is little correlation between the latents for different frames, hence the entropy model doesn't use the previous latent frame for the prediction of the target latent pixel (orange border in Fig. 14). Hence we only use a learned mask for the K highest resolution latent grids, and for the remaining grids we mask out all of the previous latent frame so that it is unused by the entropy model.

Also note that for the current latent frame τ, the relevant contexts for predicting the target latent pixel should only be a handful of values in the neighborhood of the target among the $(C^2 - 1)/2$ causal context dimensions. So it would be a waste

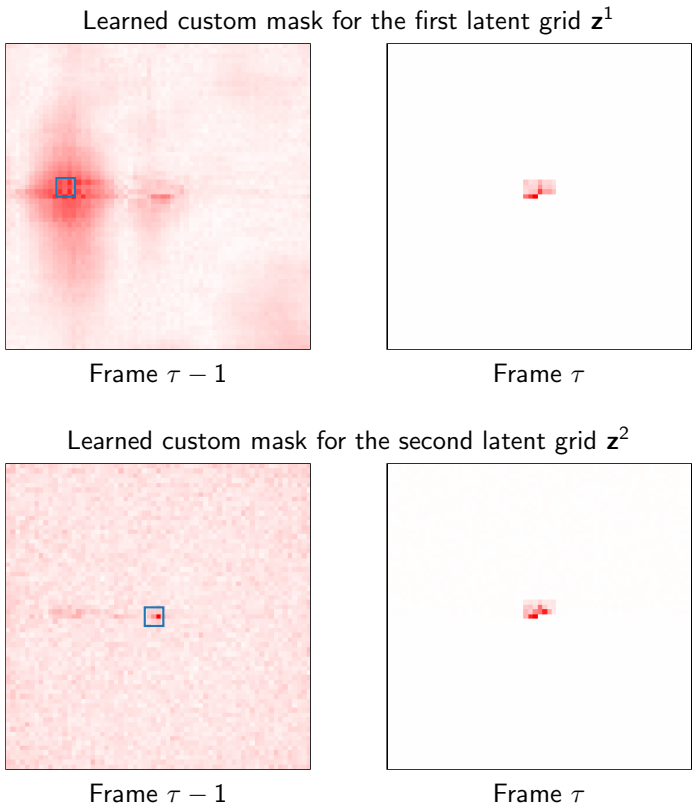

Figure 15. Visualization of the custom masks learned with the procedure described in App. A.3. The heatmap represents the magnitude of the weights in step 2 of App. A.3, and the blue box represents the learned custom masking for the previous latent frame for each latent grid.

of entropy parameters (that need to be compressed and transmitted) to process the irrelevant context dimensions. Hence we use a small causal neighbourhood of size l (where $l \leq C$ and l is odd) around the target latent pixel, the same as for images Fig. 10 (left). Note that this causal neighbourhood for the current latent frame is fixed rather than learned, and used for all latent grids rather than just the K highest resolution grids. Given this custom masking for the previous and latent frames, we train C3 from scratch, fixing the custom mask.

In practice we use $C = 65, M = 1000, c = c' = 4, K = 3, l = 7$; these values were obtained from a hyperparameter sweep on a small subset of patches of the UVG dataset.

In Fig. 15, we show that we are able to learn a sensible custom mask with the above procedure when applied to a video patch of the Jockey sequence in Fig. 13. We use a low value of RD-weight λ (Eq. (4)) for training, *i.e.*, train for a high bitrate.

The heatmaps in Fig. 15 correspond to step 2 of the procedure above (*compute magnitudes of entropy model weights for each context dimension*) when training on this video patch. After $M = 1000$ iterations, we see that the entropy model for the first latent grid $\mathbf{z}^1$ (top row) assigns the highest weights to the context dimensions that are consistent with the displacement calculated above, relative to the central target pixel. In step 3, the blue learned mask is placed here as this position has the greatest sum of magnitudes within the mask. For the second latent grid $\mathbf{z}^2$ (bottom row), we see that the region corresponding to the displacement calculated above does indeed have higher weights than its neighborhood, but the highest weight is given to the central pixel. This indicates that the correlation between the latent dimensions that correspond to the same key-point in consecutive frames is weaker for $\mathbf{z}^2$ compared to $\mathbf{z}^1$. Given that we have trained for a high bitrate, most of the information content lies in $\mathbf{z}^1$. This is consistent with the above observation that the aforementioned correlation is stronger for $\mathbf{z}^1$ compared to $\mathbf{z}^2$.

A.4. Hyperparameters

Here, we give an overview and a brief description of the hyperparameters used in our experiments as well as their settings. These comprise both architecture choices (and their hyperparameters) as well as optimization hyperparameters. As explained

in the main paper, for images we distinguish between evaluations with a single fixed setting for all images (that we simply denote as C3) and an adaptive setting where we select the best hyperparameter choice out of a small set on a per-image basis (we denote this as C3 *adaptive*). For videos we always select the best hyperparameters out of a small set on a per-patch basis. The choices of varying hyperparameters form part of the header that is transmitted with the bitstream, as they are needed to decode the image.

In Tab. 4 we provide a comprehensive list of all hyperparameters of C3. We also give their default values if they are fixed for all experiments and specify whether they are included in the *adaptive* setting.

In Tabs. 5 and 6 we separately list all hyperparameters that differ for images and videos, respectively. We provide both fixed values as well as the possible sets of values that are explored in the *adaptive* setting.

Adaptive setting for Kodak. For Kodak, the *adaptive* setting independently explores different values for three hyperparameters (see Tab. 5):

- Whether the highest resolution latent grid is included (2 choices);
- Different entropy and synthesis network sizes (3 choices);
- Different context sizes for the entropy model (2 choices)

In total the *adaptive* setting explores $2 \times 2 \times 3 = 12$ hyperparameter settings and picks the best one per image.

Adaptive setting for CLIC2020. For CLIC2020, the *adaptive* setting independently explores different values for two hyperparameters (see Tab. 5):

- Whether the highest resolution latent grid is included (2 choices);
- Different entropy and synthesis network sizes (3 choices)

In total the *adaptive* setting explores $2 \times 3 = 6$ hyperparameter settings and picks the best one per image.

Adaptive setting for UVG. For UVG, we only use the *adaptive* setting, which explores different values for six hyperparameters that are grouped together as follows (see Tab. 6). See App. D.4 for results using single/fewer settings. Namely we explore three "entropy parameter settings", (E1), (E2), (E3), that jointly specify the following hyper parameters:

- Whether a separate entropy model is learned per grid;
- Different context sizes for the entropy model;
- Whether custom masking is used

Similarly, we explore three "synthesis parameter settings", (S1), (S2), (S3), that jointly specify the following hyperparameters:

- Whether the bias of the last layer of the synthesis network is initialized to the mean RGB values of the image;
- Whether the $3 \times 3 \times 3$ 3D convolutions in the synthesis model are replaced by 3×3 2D convolutions per frame.

In total the adaptive setting explores $3 \times 3 = 9$ hyperparameter settings and picks the best one per video patch.

Moreover, the video patch size is chosen according to the RD-weight λ, as we observed that larger patches give better RD performance for high values of λ (low bitrates) and vice versa. We use $(30 \times 180 \times 240)$ for $\lambda \leq 2 \cdot 10^{-4}$, $(60 \times 180 \times 240)$ for $2 \cdot 10^{-4} < \lambda \leq 10^{-3}$, $(75 \times 270 \times 320)$ for $\lambda > 10^{-3}$.

Hyperparameter	**Fixed value**	**In *adaptive* setting?**
Quantization – Stage 1		
Number of encoding iterations	10^5	
Initial learning rate	10^{-2}	
Final learning rate	0	
Initial value of T for soft rounding	0.3	
Final value of T for soft rounding	0.1	
Initial value of a for Kumaraswamy noise	2.0 (📷) / 1.75 (📹)	
Final value of a for Kumaraswamy noise	1.0	
Threshold for gradient L2 norm clipping	0.1 (📷) / 0.03 (📹)	
Quantization – Stage 2		
Maximum number of encoding iterations	10^4	
Initial learning rate	10^{-4}	
Decay lr if loss has not improved for this many steps	20	
Decay lr by multiplying with this factor	0.8	
Finish Stage 2 early if lr drops below this value	10^{-8}	
Value of T for soft-rounding gradient estimation	10^{-4}	
Architecture – Latents		
Number of latent grids	–	
Quantization step (bin width) for rounding	–	
Use the highest resolution grid $(\{t\}, h, w)$?	–	📷
Architecture – Synthesis model		
Output channels of the 1×1 convolutions (list)	–	📷 and 📹
Number of 3×3 residual convolutions (with 3 channels)	2	
Initialize the last layer bias with mean RGB of the image?	–	📹
(video only) Replace $3 \times 3 \times 3$ Conv with 3×3 Convs per frame	–	📹
Architecture – Entropy model		
Widths of the 1×1 convolutions	–	📷 and 📹
Log-scale of Laplace is shifted by ... before exp	8	
Scale parameter of Laplace is clipped to	$[10^{-3}, 150]$	
Context size (same grid)	–	📷 and 📹
Include previous grid in context?	–	📷
Context size (previous grid)	3×3	
Architecture – Entropy model (video only)		
Learn separate models per grid, $\psi = (\psi_1, \dots \psi_N)$?	–	📹
Use custom masking (cf. App. A.3)?	–	📹
Mask size (current frame), l	7×7	
Mask size (previous frame), $c \times c'$	4×4	
Iteration count to learn the mask, M	1000	
Number of grids for which mask is learned, K	3	
Other		
Possible quantization steps for network parameters	♣	
(video only) Size of video patches	–	📹

Table 4. Hyperparameters and their values for the quantization-aware optimization and architecture of C3 for images (📷) and video (📹). Where hyperparameters are fixed for all experiments and evaluations, their values are listed. Otherwise they are specified in the image or video specific hyperparameter list in Tabs. 5 and 6, respectively. An icon in the last column indicates whether a hyperparameter is included in the *adaptive* setting for images and/or videos. Except for gradient L2 norm clipping, all quantization hyperparameter values are the same for images and videos. ♣ The possible quantization steps for the network parameter θ and ψ are $\{5 \cdot 10^{-5}, 10^{-4}, 5 \cdot 10^{-4}, 10^{-3}, 3 \cdot 10^{-3}, 6 \cdot 10^{-3}, 10^{-2}\}$ for the weights and biases separately.

Hyperparameter	**Fixed value**	**Adaptive values**
Number of latent grids	7	–
Latent quantization step (bin width) for rounding	0.4	–
Use the highest resolution grid (h, w)?	✓	✓, ✗
Initialize the last synthesis layer bias with mean RGB of the image?	✗	–
Learn separate entropy model per grid, $\psi = (\psi_1, \dots \psi_N)$?	✗	–
Kodak only		
Include previous grid in context?	✗	–
Output channels of synthesis 1×1 convs♣	$(18, 18)$	$(12, 12), (18, 18), (24, 24)$
Output channels of entropy 1×1 convs♣	$(18, 18)$	$(12, 12), (18, 18), (24, 24)$
Context size (same grid) in entropy model	7×7	$5 \times 5, 7 \times 7$
CLIC2020 only		
Include previous grid in context?	✓	–
Output channels of synthesis 1×1 convs♣	$(24, 24)$	$(12, 12), (18, 18), (24, 24)$
Output channels of entropy 1×1 convs♣	$(24, 24)$	$(12, 12), (18, 18), (24, 24)$
Context size (same grid) in entropy model	7×7	–

Table 5. Hyperparameter values that are specific to images. *Fixed value* contains the values that are fixed for all images. It also contains the default values in the (non-adaptive) setting. *Adaptives values* specifies the values that are explored in the adaptive setting.
♣ The adaptive values for the synthesis and entropy model sizes are varied together.

Hyperparameter	Fixed value	Adaptive values					
		E1	E2	E3	S1	S2	S3
Size of video patches	–	$(30 \times 180 \times 240), (60 \times 180 \times 240), (75 \times 270 \times 320)$♣					
Number of latent grids	–	6	5	6			
Latent quantization step (bin width) for rounding	0.3						
Use the highest resolution grid (t, h, w)?	✓						
Learn separate entropy model per grid, $\psi = (\psi_1, \dots \psi_N)$?	–	✗	✓	✓			
Output channels of entropy 1×1 convs	–	$(16, 16)$	$(2, 2)$	$(8, 8)$			
Context size (same grid) in entropy model	–	$3 \times 9 \times 9$	$3 \times 9 \times 9$	$3 \times 65 \times 65$			
Use custom masking?	–	✗	✗	✓			
Include previous grid in context?	✗						
Output channels of synthesis 1×1 convs	$(32, 32)$						
Initialize the last synthesis bias with mean RGB of the image?	–				✓	✓	✗
Replace $3 \times 3 \times 3$ Conv with 3×3 Convs per frame	–				✓	✗	✓

Table 6. Hyperparameter settings that are specific to videos. *Fixed value* contains the hyperparameter values that are fixed for all images. *Adaptive values* lists the hyperparameter values that are explored for each patch separately. There are three possible settings for the entropy model, (E1), (E2), (E3), and three possible settings for the synthesis model, (S1), (S2), (S3). Therefore, for each patch, we explore $3 \times 3 = 9$ different settings.

♣ The video patch size is chosen according to the RD-weight λ. We use $(30 \times 180 \times 240)$ for $\lambda \leq 2 \cdot 10^{-4}$, $(60 \times 180 \times 240)$ for $2 \cdot 10^{-4} < \lambda \leq 10^{-3}$, $(75 \times 270 \times 320)$ for $\lambda > 10^{-3}$.

B. Evaluation details

B.1. BD-rate

The Bjøntegaard Delta rate (BD-rate) metric [9] is a scalar that estimates the saving in bitrate of one RD curve compared to another. It is useful since it allows to quantify the improvement of one RD curve over another with a single scalar. Given an anchor RD curve and a candidate RD curve, the curves are first transformed into a curve of distortion vs log-rate. Then the difference between the area under the curve (with respect to the distortion/PSNR axis) of the candidate and the anchor are computed. Note that since the area under the curve is measured with respect to the distortion axis, the smaller the area under the curve, the better. Therefore a negative value of the BD-rate implies that the candidate curve is better than the anchor. Also note that the output is invariant to (positive) scalar multiplication of the rate, due to the use of the log rate. Hence the BD rate should be invariant to whether we use bpp, bits or nats.

B.2. PSNR evaluation for videos

We compute PSNR with the following convention also used in Mentzer et al. [60]: take the per-frame PSNR for each frame of a given video, then take the mean across all frames for that video to get a PSNR value for that video. Take the mean of these values across all videos to get the PSNR for a given RD-weight. For bpp, simply take the mean across all patches of a video to get the bpp for a given video, then take the mean across all videos.

B.3. Entropy coding

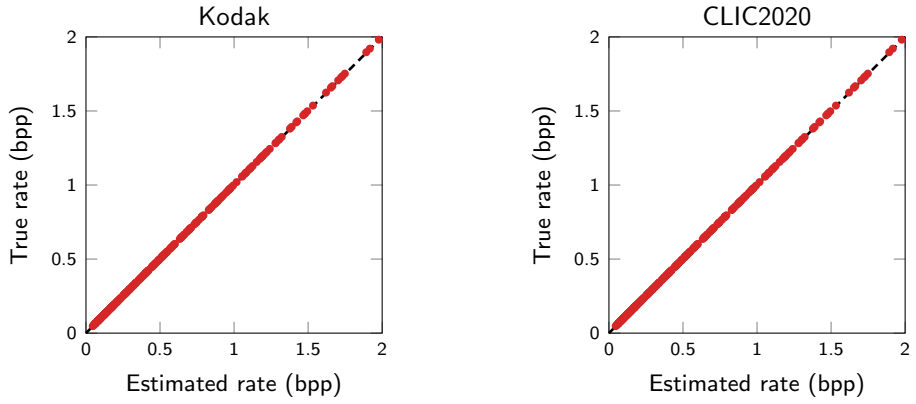

Figure 16. The true bit-rate observed when range coding images into a bit-stream is nearly identical to our estimates of the bit-rate obtained by evaluating log-probabilities. Each point corresponds to an image encoded at a given rate-distortion weight.

To encode images to files, we use the `range-coder` Python package available in PyPI. First, hyperparameter choices and other information that depend on the image are uniformly encoded. These currently include the image resolution and the choice of quantization step widths for the synthesis transform and the entropy model. Other hyperparameters (such as the model architecture) are assumed to be fixed and known to the decoder. Next, the weights and biases of the synthesis transform and entropy model are encoded assuming a different zero-mean Laplace distribution for each layer's set of weights and set of biases. The scale of the Laplace distribution is estimated from the parameters themselves, quantized to one of 1024 possible values, and uniformly entropy encoded using the range coder. Finally, the latent grids are autoregressively encoded with a Laplace distribution whose mean and scale are predicted by the entropy model.

For convenience and faster turnaround times, bpp values used in plots throughout the paper were estimated by evaluating the log-probabilities that would be used when range coding model parameters and latent values. We find that these estimates are very close to the bit-rates observed when range coding using the default, non-adaptive setting of C3 (Fig. 16).

B.4. Estimating the decoding complexity in MACs

Here we explain in more detail how we obtained the estimates for the decoding complexity. We report this value in terms of the number of multiply-accumulate operations (MACs) per pixel. We follow the convention that $1\,\text{MAC} = 2\,\text{FLOPs}$, though note that this is not always consistently done in the literature.

We follow Cool-chic [43] and Cool-chic v2 [48] and report theoretic MACs for applying the neural networks in our model, *i.e.*, matrix multiplications, but exclude pointwise operations such as non-linearities. We also include an estimate for bi-/trilinear upsampling.

Cool-chic v2 uses `fvcore` [22] to automatically estimate the number of theoretic MACs in `PyTorch` [66]. We implement our method in `JAX` [10], which (to the best of our knowledge) does not easily allow for the automatic estimation of theoretic MACs/FLOPs. We therefore use back-of-the-envelope estimates as detailed below and confirm they agree with the numbers as reported by `fvcore` in a `PyTorch` codebase.

The MACs estimates for other baselines were obtained from various sources while ensuring the numbers were comparable with ours. For BMS [6], MBT [64] and CST [16], we calculated the MACs using the `fvcore` library on the CompressAI implementations of these models (measuring MACs only on the decoder of the model). For MLIC and MLIC+ [38], the numbers were directly provided to us by the authors, who calculated their numbers using the `DeepSpeed` library. For EVC [29], we used the numbers for the EVC-S, EVC-M and EVC-L models in the paper, which were obtained using the `ptflops` library.

Bi- and trilinear upsampling To upsample the latent grids we use bilinear upsampling for images and trilinear upsampling for videos, respectively. `fvcore` does not provide a complexity estimate for upsampling; we therefore use the following upper-bound estimates. We upper bound the complexity of bilinear upsampling with 8 MACs per output pixel; this estimate includes computing the weighted average of the values at the four closest grid points (4 MACs) and computation of the corresponding weights (4 MACs). Similarly, we upper bound the complexity of trilinear upsampling with 16 MACs per output pixel; this estimate includes computing the weighted average of the values at the eight closest grid points (8 MACs) and computation of the corresponding weights (8 MACs). Note that in practice, most of the weights can be pre-computed and cached, especially when we are upsampling by an exact factor of 2.

Application of the entropy model. While we implement the entropy as a convolutional network with a masked convolution as first layer and a sequence of 1×1 convolutions during encoding, it can be equivalently evaluated as a feed-forward MLP where the context for each latent grid location is read from memory. Each latent grid location is evaluated independently in this case. Therefore, the cost of evaluating the entropy model on all latent entries is given by the cost of applying it to one entry multiplied by the number of latent values $\mathcal{Z}$, which is given by

$$\mathcal{Z} = \sum_n h_n \cdot w_n \tag{25}$$

where h_n and w_n are the height and width of latent grid n. The cost of applying the entropy model is the cost of applying all layers, which depends on the input, hidden, and output sizes. The cost in MACs of each single layer is simply the product of the input and output size, $c_{\text{in}} \cdot c_{\text{out}}$. The total complexity in MACs/pixel is then given by adding the contributions from all latent values and dividing them by the number of pixels.

We verify that the numbers we obtain for the application of the feed-forward MLP are the same as those reported by `fvcore` that is used by Cool-chic v2. When estimating the cost of additionally conditioning on the previous grid, we need to take into account the larger input-size to the entropy network as well as the bilinear upsampling of the latent. We upper-bound the cost of the bilinear upsampling by $\mathcal{Z} \cdot \text{cost}_{\text{one upsampling}}$ as we have to resample exactly one value from a previous grid for each current grid location. This is an upper bound because the first grid does not actually depend on a previous grid.

Application of the synthesis model The synthesis model f_θ is a simple convolutional network with skip connections in its second part. The cost of applying a single convolutional layer at one location is

$$k \cdot k \cdot c_{\text{in}} \cdot c_{\text{out}}. \tag{26}$$

As the number of locations is given by the number of pixels, the above estimate is the cost of applying a single layer in MACs/pixel. We again evaluate the cost for different network sizes and verify that they agree with the numbers reported by `fvcore` when implementing them.

C. Additional results

C.1. Full RD curves with all baselines

We include several large-format RD-curve plots in which we compare C3 to:

1. overfitted neural compression methods on the Kodak image benchmark in Fig. 18,
2. autoencoder based neural compression methods on the Kodak image benchmark in Fig. 19,

Component	KODAK	CLIC	UVG
Entropy model	1600	1889	2540
Upsampling	48	48	80
Synthesis model	978	978	1798
Total	2626	2925	4418

Table 7. Maximum computational complexity in MACs/pixel of different components of C3 for the hyperparameters used in each dataset.

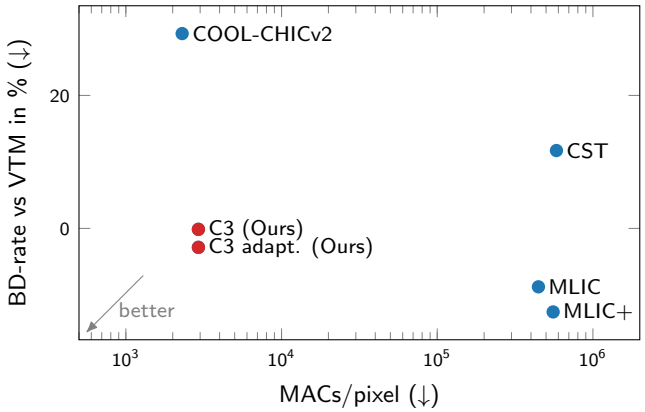

Figure 17. MACs/pixel vs BD-rate on CLIC2020. C3 performs well both in terms of BD-rate and decoding MACs/pixel, achieving a better trade off than existing neural codecs.

3. several additional baselines on the CLIC2020 image benchmark in Fig. 20, and
4. several additional baselines on the UVG video benchmark in Fig. 21.

Image baselines. In addition to the baselines used in the main paper, for images we also compare against[1]: COIN [20], SPY [76], COIN++ [21], MSCN [71], VC-INR [72], RECOMBINER [32], MBT [64], ELIC [31] and STF [90]. Cool-chic, Cool-chic v2, COIN, COIN++ and STF results were obtained from official code implementations. MLIC, SPY, MSCN, VC-INR and RECOMBINER results were obtained from direct communication with the respective paper authors. BPG, VTM, BMS, MBT, CST and ELIC were obtained from CompressAI [7].

Video baselines. In addition to the baselines cited in the main paper, for videos we also include HEVC (HM 18.0, Random Access, default setting) [77], VTM (17.0, Random Access, default setting) [11], Insta-SSF18 [81] (bigger model than Insta-SSF5), DCVC [49], ELF-VC [70], NeRV [14] and HNeRV [15]. HEVC (RA), NeRV, HNeRV, FFNeRV and HiNeRV results were obtained from Kwan et al. [42] or direct communication with the authors. HEVC (medium, no B-frames), DCVC, ELF-VC and VCT results were obtained from direct communication with the authors of Mentzer et al. [60]. Insta-SSF5/18 and MIMT results were obtained directly from their papers. VTM (RA) results were obtained by running VTM 17.0 using the default setting of `encoder_randomaccess_vtm.cfg` [84].

Discussion of the additional results. From Fig. 18, we see that on the Kodak image benchmark, C3 is SOTA among methods based on overfitting neural fields to a single image instance by a noticeable margin, and is the only method that is competitive with VTM . From Fig. 19, we see that C3 is not as competitive as the more recent autoencoder-based methods in terms of RD, although the gap has been significantly reduced compared to Cool-chic v2.

On CLIC2020, arguably a more realistic dataset than Kodak with images at higher resolution, we see in Fig. 20 that C3 *adaptive* outperforms VTM and is quite close to MLIC+, the best-performing baseline. We also show the plot of BD-rate against decoding complexity in Fig. 17; C3 performs much better in terms of RD performance compared to other baselines with a low decoding complexity such as Cool-chic v2.

On UVG, we show in Fig. 21 that C3 is also a competitive baseline for video compression, despite being the first one of its kind (in the Cool-chic line of work) to be applied to videos. Note that there is a clear room for improvement that is visible in

[1]We follow the CompressAI [7] convention of using the first letters of the first three authors for unnamed methods

the gap between C3 and stronger baselines such as HiNeRV and MIMT, however we emphasize again that C3 achieves its competitive performance with 2-3 orders of magnitude lower decoding complexity than these baselines (cf. Fig. 9).

C.2. Encoding times

See Tab. 8 for details on encoding times for each dataset, showing the fastest and slowest settings among the hyperparameter settings in the adaptive sweeps. Note that the encoding time for CLIC2020 depends on the size of the image, hence we measure it on the largest image of resolution 1370×2048. We emphasize again that we do not optimize for encoding times and use unoptimized research code to obtain these encoding times.

Hyperparameter	**Value**	**Encoding time (sec/1k steps)**
Kodak – fastest setting		3.9
Context size (same grid)	5×5	
Width of 1×1 convolutions (synthesis & entropy)	$(12, 12)$	
Use the highest resolution grid (t, h, w)?	✗	
Kodak – slowest setting		7.1
Context size (same grid)	7×7	
Width of 1×1 convolutions (synthesis & entropy)	$(24, 24)$	
Use the highest resolution grid (t, h, w)?	✓	
CLIC2020 – fastest setting		21.5
Width of 1×1 convolutions (synthesis & entropy)	$(12, 12)$	
Use the highest resolution grid (t, h, w)?	✗	
CLIC2020 – slowest setting		48.0
Width of 1×1 convolutions (synthesis & entropy)	$(24, 24)$	
Use the highest resolution grid (t, h, w)?	✓	
UVG – fastest setting		28.7
Patch size	$(30, 180, 240)$	
Entropy setting	No conditioning	
Replace $3 \times 3 \times 3$ Conv with 3×3 Convs per frame	✓	
UVG – slowest setting		456.7
Patch size	$(75, 270, 320)$	
Entropy setting	Learned mask	
Replace $3 \times 3 \times 3$ Conv with 3×3 Convs per frame	✗	

Table 8. Encoding times for C3 measured on a single NVIDIA V100 GPU.

C.3. RD curves for individual UVG videos

In Fig. 22, we include RD curves for individual UVG videos. Note that C3 tends to perform better than VCT for Beauty, Bosphorus, Honeybee, Shakendry and Yachtride, and even outperforms VTM (17.0, random access setting) on Beauty and Shakendry. However C3 appears to struggle with Jockey and Readysetgo, which are the video sequences with faster motion. While we show in App. D.4 that the learned mask helps to achieve a better RD performance, it would be interesting to investigate how the performance can be improved further especially on these sequences with fast motion.

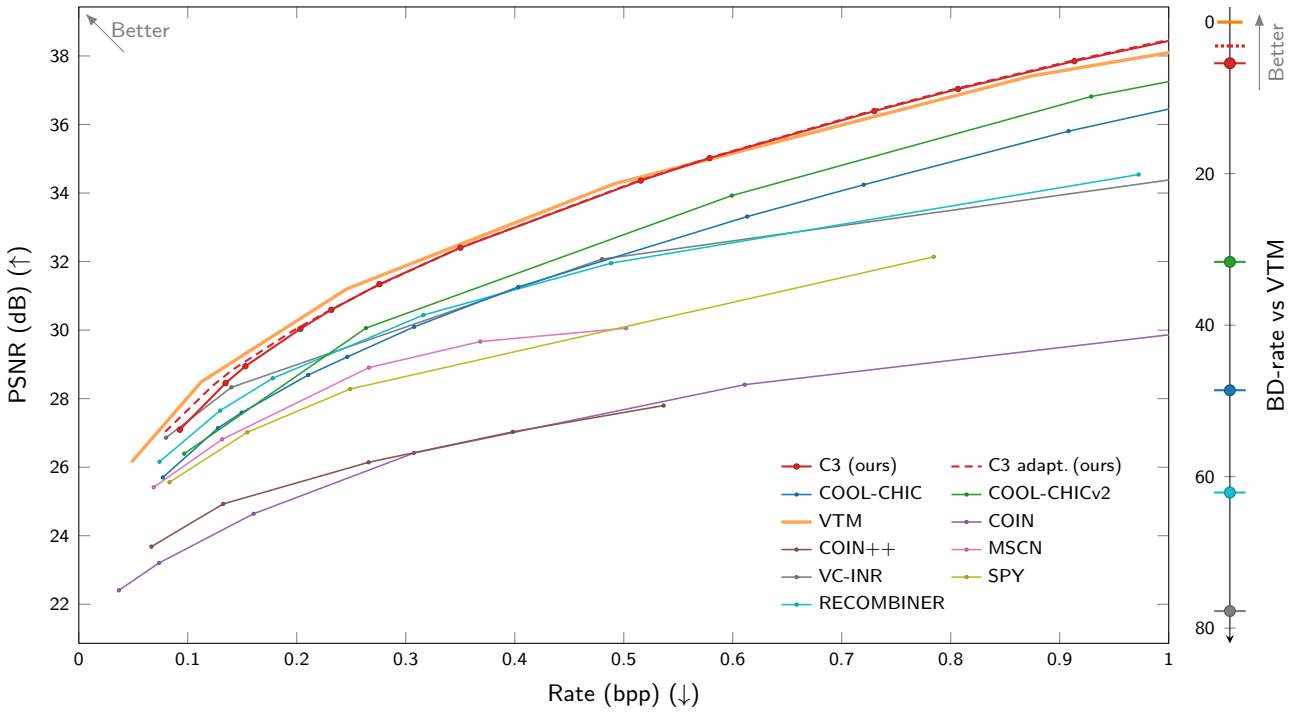

Figure 18. Rate-distortion curve and BD-rate on the Kodak image benchmark comparing C3 to other overfitted neural field based compression methods, including those in Fig. 5. Note that we omit methods with very large values from the BD-rate plot on the right.

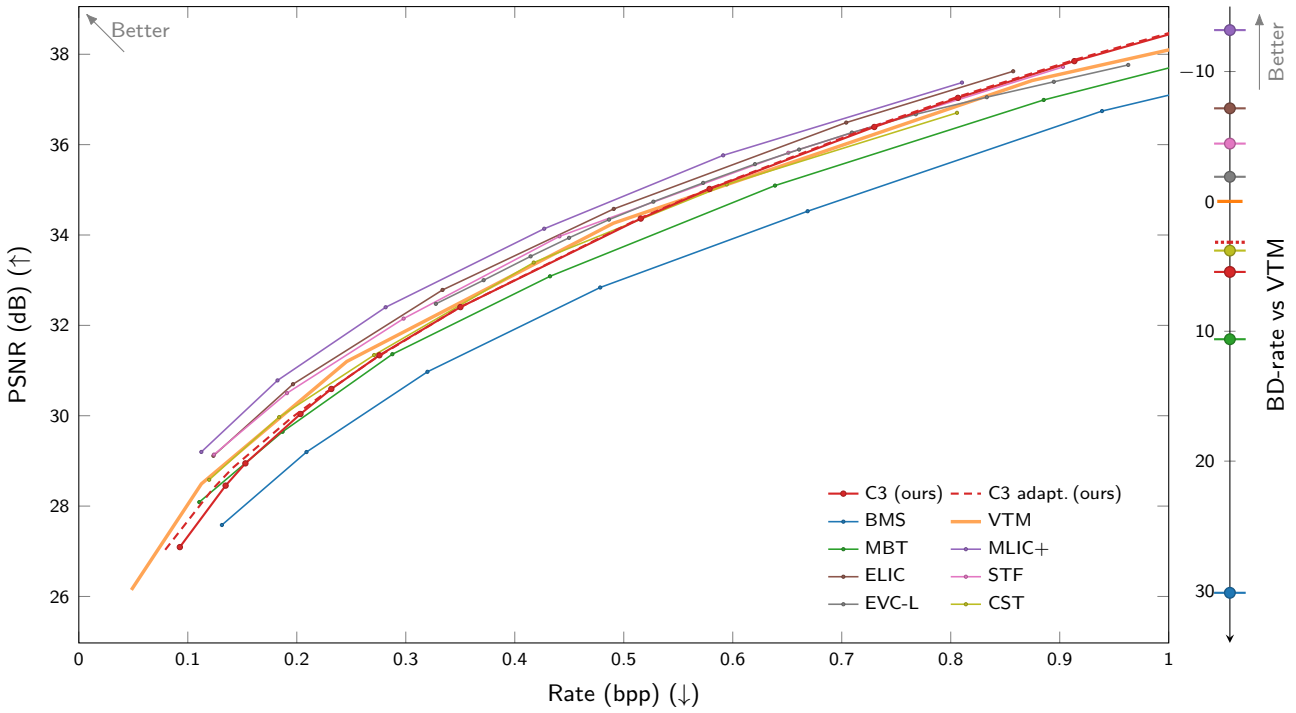

Figure 19. Rate-distortion curve and BD-rates on the Kodak image benchmark comparing C3 to autoencoder-based neural compression methods, including those in Fig. 5.

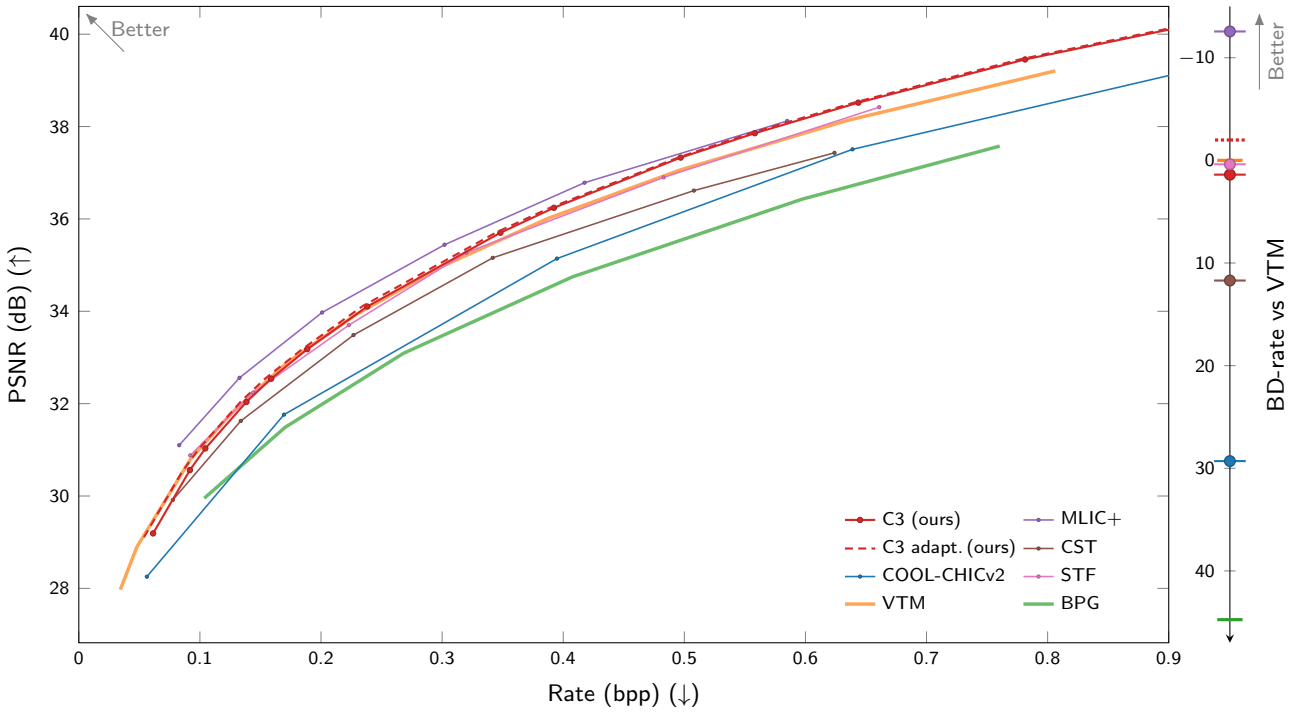

Figure 20. Rate-distortion curve and BD-rates of more baselines on CLIC2020, including those in Fig. 6.

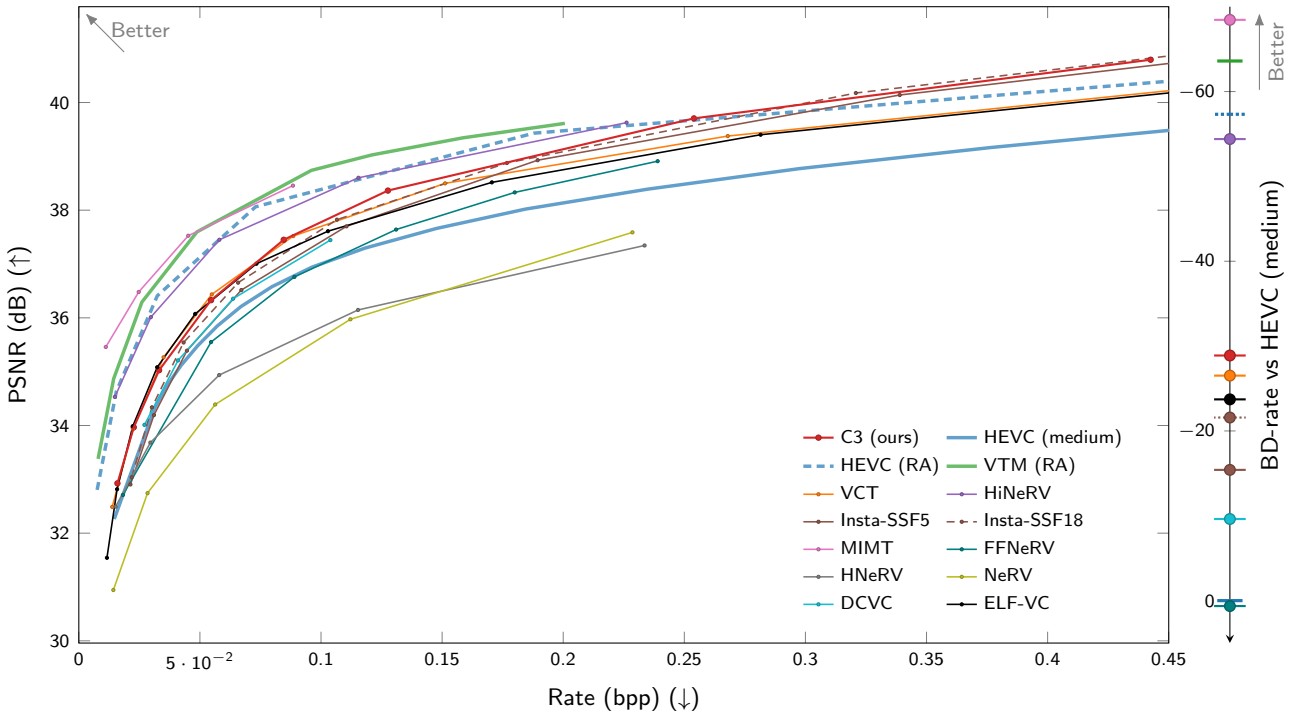

Figure 21. Rate-distortion curve of more baselines on all UVG videos, including those in Fig. 8.

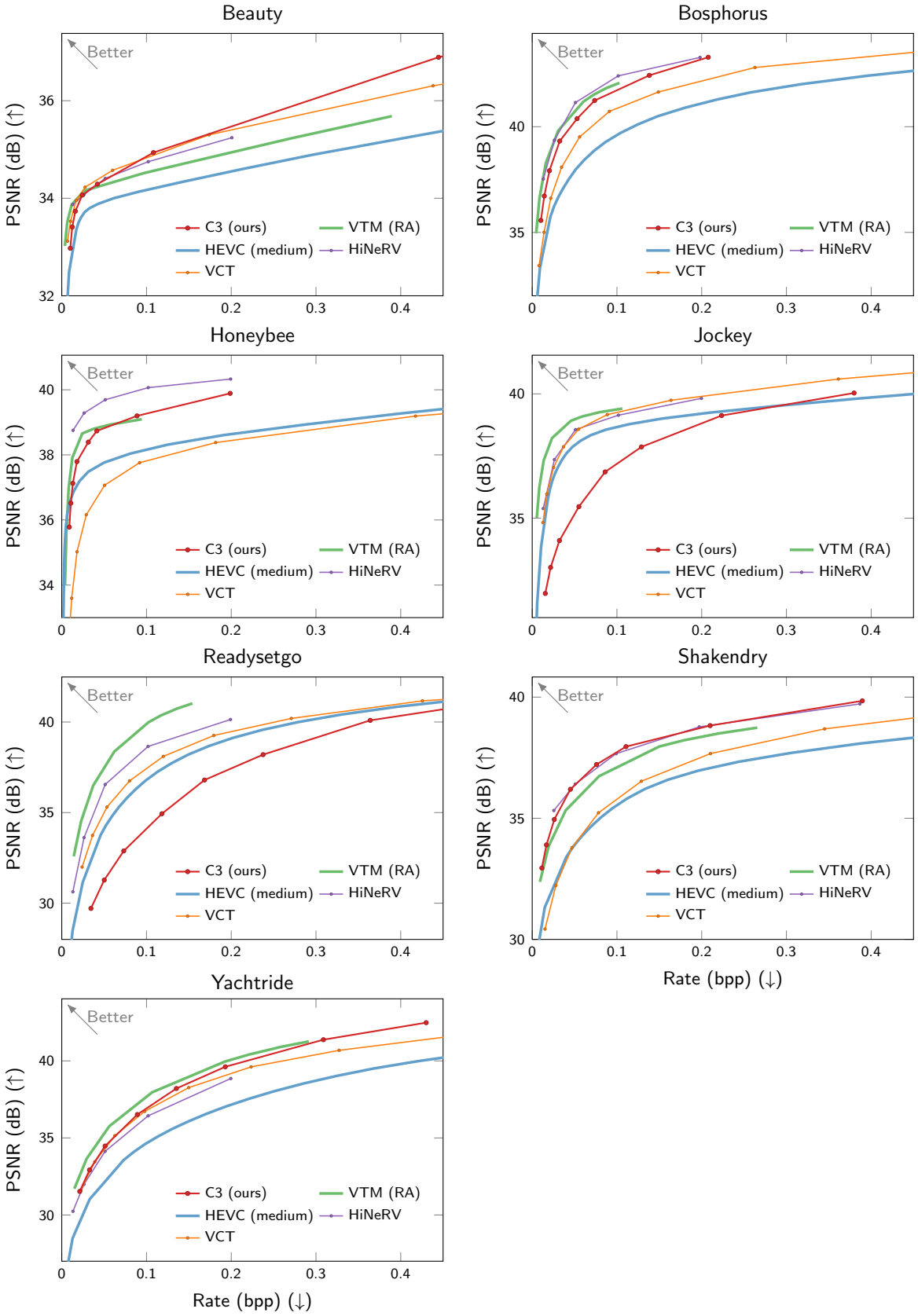

Figure 22. Rate-distortion curves for individual videos in UVG.

D. Additional ablations

D.1. CLIC2020 ablations

In Tab. 9 we show the ablations for CLIC2020 when sequentially removing each of our improvements from the best performing C3 Adpative model, similarly to Tab. 2 for Kodak in the main paper. In Tab. 10, we show the results when disabling individual features from the C3 model, similar to Tab. 3 for Kodak in the main paper. We have an additional row for the previous grid conditioning described in App. A.1.4, since this option is only used for CLIC2020. The conclusions are similar in that soft-rounding, the GELU activation function and Kumaraswamy noise account for most of the boost in RD performance. Two minor differences for CLIC2020 are: 1) the quantization step is slightly more important than the remaining features compared to Kodak, 2) conditioning on the previous grid improves performance for CLIC2020. In Tab. 11 we ablate the annealing schedules for the soft-rounding temperature and the shape parameter of the Kumaraswamy noise. We find that fixing them to a single value leads to worse performance. A lower soft-rounding temperature results in a closer approximation of quantization but higher variance in the gradients. Annealing the temperature allows us to control this bias-variance trade-off over time, with less biased gradients becoming more important later in training. Also note that different shape parameters of the noise are optimal for different soft-rounding temperatures.

Model variant	**BD-rate vs. C3 Adaptive**
C3 (adaptive)	0.0%
C3	3.2%
✗ Adaptive lr (Stage 2)	4.6%
✗ Shifted log-scale	4.9%
✗ Quantization step < 1	7.2%
✗ Previous grid conditioning	8.8%
✗ Kumaraswamy noise	20.2%
✗ GELU	45.4%
✗ Soft-rounding	65.3%

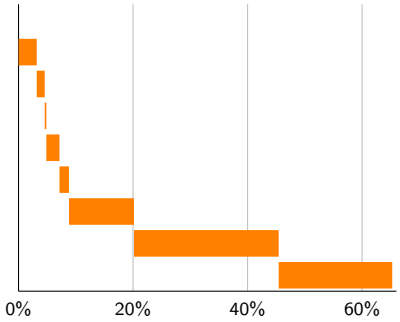

Table 9. CLIC2020 ablation sequentially removing methodological changes. Note that a higher BD-rate means worse RD performance relative to our default settings for C3.

Removed Feature	**BD-rate vs. C3**
C3 − Soft-rounding	24.87%
C3 − GELU	9.83%
C3 − Kumaraswamy noise	8.06%
C3 − Quantization step < 1	2.82%
C3 − Previous grid conditioning	1.50%
C3 − Shifted log-scale > 0	0.99%
C3 − Adaptive lr (Stage 2)	0.54%

Table 10. CLIC2020 ablation knocking out individual features from C3 (fixed hyperparameters across all images). Note that a higher BD-rate means worse RD performance relative to our default settings for C3.

D.2. Effect of Stage 2

We also ablate the increase in performance that we can attribute to stage 2 of optimization, by comparing the default setting of C3 (both stage 1 + 2) vs only having stage 1. We find that the BD-rate with respect to VTM for these two settings is $+1.39\%$ vs $+2.00\%$ on the CLIC2020 benchmark. The gain in BD-rate for stage 2 is indeed not as significant as was observed for Cool-chic v2 [48].

Soft-round temperature T	Kumaraswamy noise a	BD-rate vs. C3
0.1	1.0	88.48%
0.1	1.5	109.08%
0.1	2.0	129.96%
0.2	1.0	11.66%
0.2	1.5	5.66%
0.2	2.0	5.97%
0.3	1.0	10.32%
0.3	1.5	5.57%
0.3	2.0	9.78%

Table 11. Ablation of annealing schedules for the soft-rounding temperature and the shape parameter of the Kumaraswamy noise on CLIC2020. Instead of annealing the soft-rounding temperature from 0.3 to 0.1 and the Kumaraswamy noise parameter from 2 to 1, we clamp them at the fixed value specified in the table. A higher BD-rate corresponds to worse RD performance relative to C3 with annealing.

D.3. BD-rate vs encoding iterations/time evaluation for CLIC and UVG

In Fig. 23 we show how the BD-rate of C3 changes as a function of the number of encoding iterations for both CLIC2020 and the Shakendry sequence of UVG. Note that with around $20 - 30$k iterations, we can approach the BD-rate of the default setting (100k iterations).

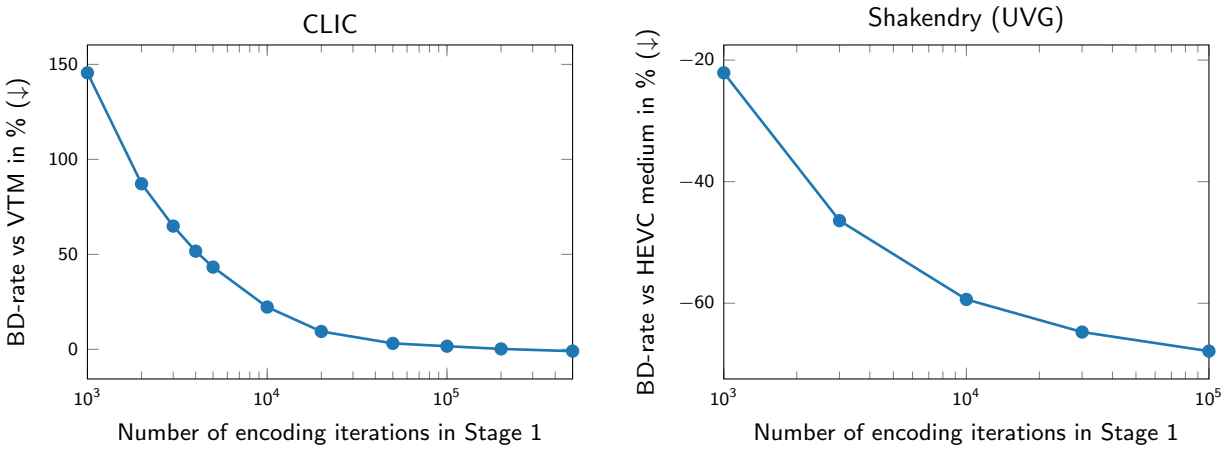

Figure 23. Ablation on the effect of using different number of encoding iterations in Stage 1 of the optimization. We evaluate the BD-rate (right) vs VTM (lower BD-rate values are better) on CLIC2020 and (left) vs HEVC on the Shakendry sequence of UVG. The number of encoding iterations in Stage 2 is determined adaptively but set to be at most 10% of the number of iterations in Stage 1.

D.4. Video ablations

Settings	BD-rate vs. HEVC (medium)
Single setting: E1 S1	−7.67%
3 settings: single setting for S1 but sweep E1, E2, E3	−21.44%
9 settings (default): sweep all combinations of E1, E2, E3 and S1, S2, S3	−28.89%

Table 12. UVG ablation for the 9 hyperparameter settings used (cf. App. A.4). Note that lower BD-rate means better RD performance.

In Fig. 24, we highlight the importance of learning the mask by comparing the rate and distortion values when using different custom mask locations for the previous latent grid. Among the four different choices of mask locations, we see that

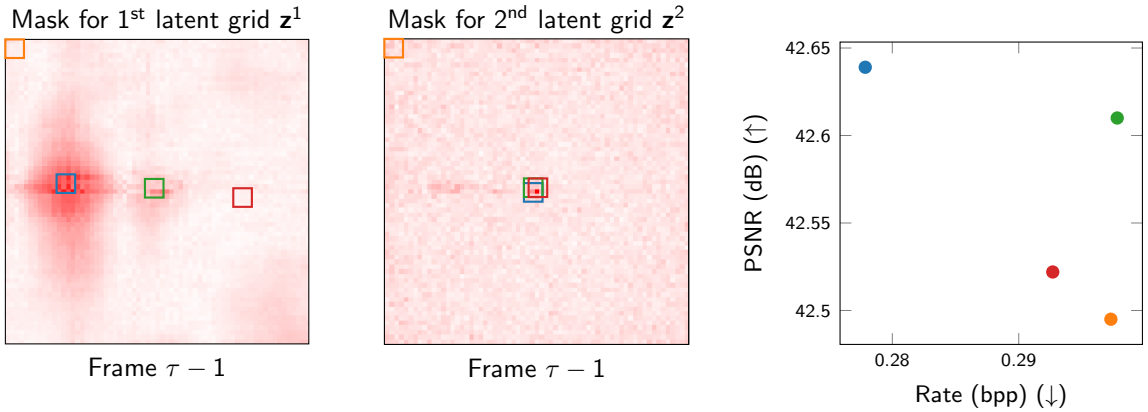

Figure 24. Comparison of (bpp, psnr) when training with different mask locations for the previous latent frame on the Jockey patch in Fig. 13. The different locations are colour coded as follows: learned (——), top-left (——), center (——), diametrically opposed to learned (——),

the learned mask (same as mask shown in Fig. 15) achieves the best RD values.

In Tab. 12, we show an ablation for how the BD-rate changes when we use a subset of the 9 settings used for the video experiments on the UVG dataset (see Tab. 6 for details on the 9 settings). We see that varying the entropy settings and the synthesis settings are both important for improvements in performance.

E. Additional visualizations

In this section, we show visualizations of C3 reconstructions and latents for both images and video.

Figure 25. Qualitative comparison between C3 (top) and Cool-chic v2 (bottom). The PSNR for C3 is 30.28dB and the PSNR for Cool-chic v2 is 28.98dB.

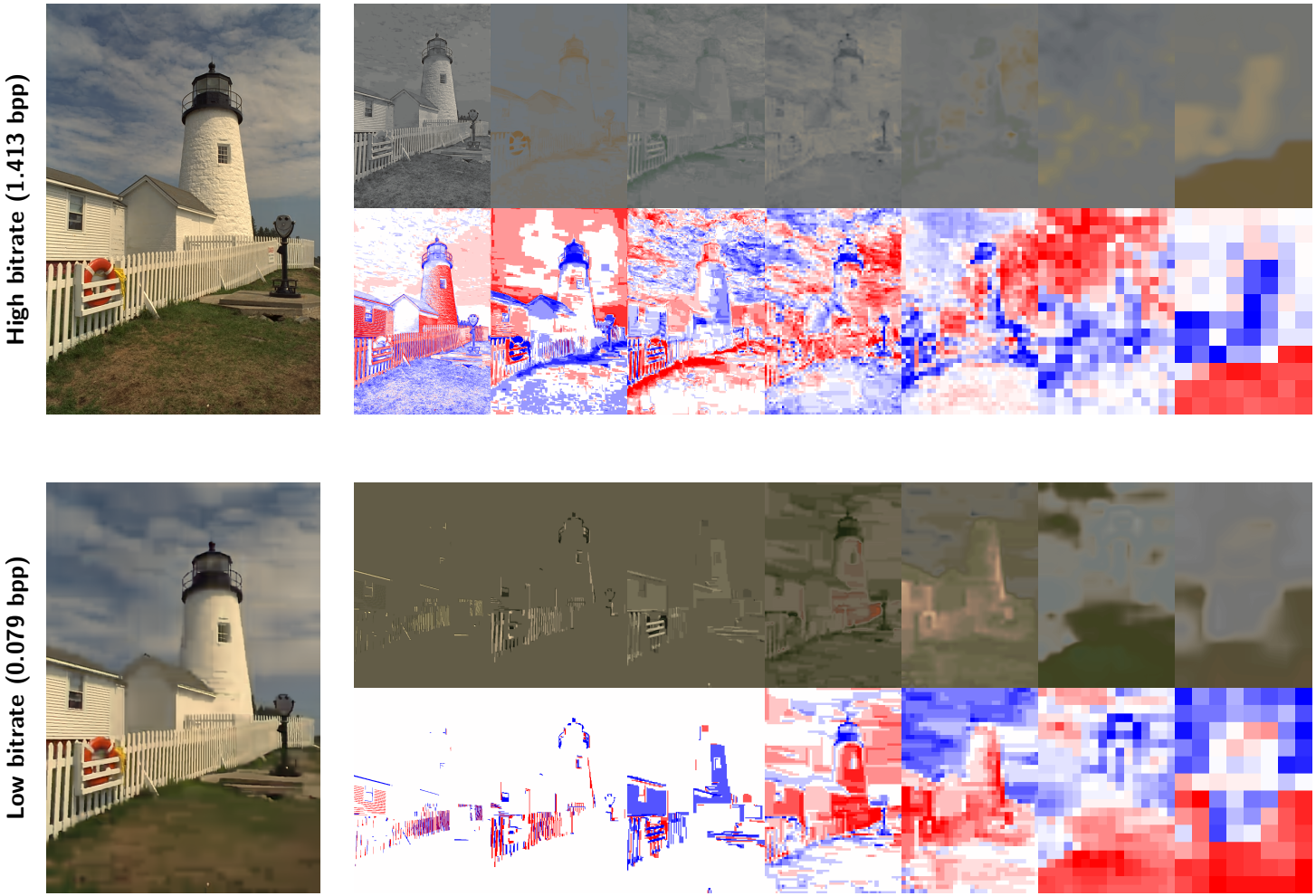

Figure 26. *Top:* Reconstruction and visualization of C3's latents for kodim19 at a high bit-rate (1.413 bpp). The first row shows reconstructions when all but one out of 7 sets of latents is set to zero. For example, the highest resolution latent grid appears to encode luminance information. The second row visualizes the raw latents, upscaled to match the resolution of the output. *Bottom:* As above but at a lower rate (0.079 bpp).

Ground Truth

HNeRV [15]
39.1dB PSNR@0.101bpp

HiNeRV [42]
41.1dB PSNR@0.051bpp

C3 (ours)
40.4dB PSNR@0.054bpp

Figure 27. Reconstructions of a frame of Bosphorus from the UVG dataset for various models. Adapted from Figure 9 of Kwan et al. [42].

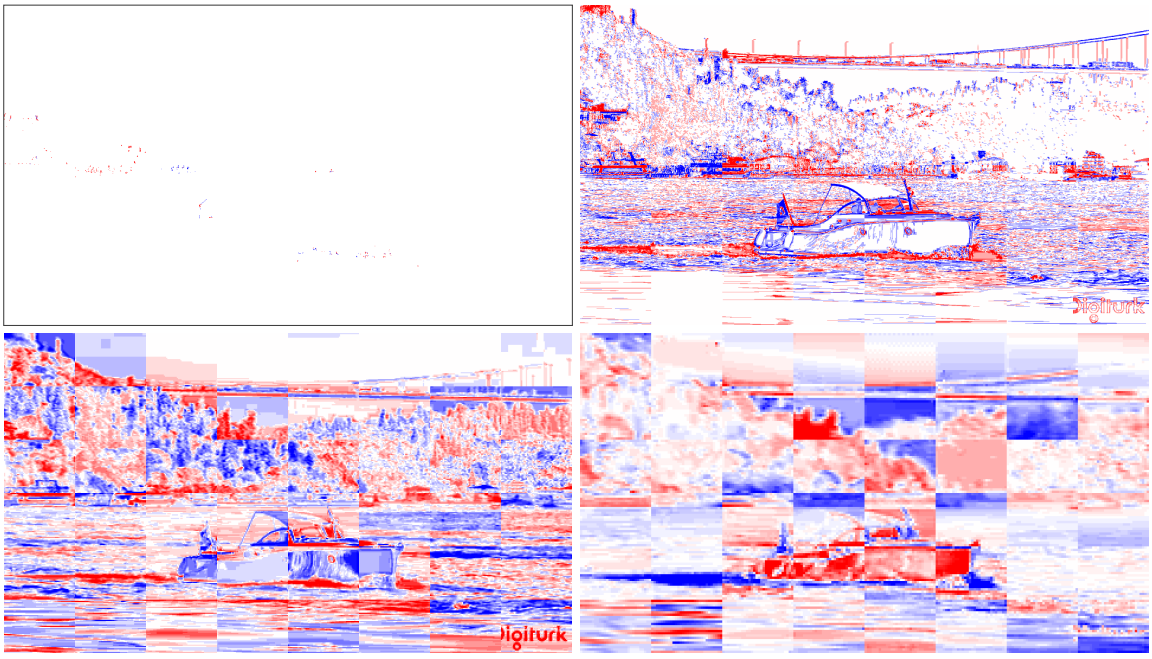

Figure 28. C3 latents of the first four grids corresponding to the frame in Fig. 27. Note that the highest resolution grid (top left) are mostly zeros, hence highly compressible. Also note that we see patch artifacts in the latents because each patch has been optimized independently, so we have different parameter values for the synthesis and entropy models of each patch. However we see in Fig. 27 that these artifacts are not visible in reconstructions even for bpp values around 0.05.

F. Raw values

In this section we provide the raw values for the RD-curves of C3 on all benchmarks.

Rate [bits per pixel]	PSNR [dB]
0.0927	27.092
0.1347	28.453
0.1529	28.946
0.2034	30.038
0.2317	30.592
0.2758	31.338
0.3501	32.404
0.5156	34.364
0.5787	35.019
0.7299	36.391
0.8067	37.035
0.9134	37.848
1.0789	38.977
1.4087	40.841

(a) C3

Rate [bits per pixel]	PSNR [dB]
0.0793	27.032
0.1228	28.377
0.1409	28.839
0.1947	29.946
0.2252	30.489
0.2718	31.267
0.3490	32.392
0.5146	34.369
0.5783	35.026
0.7277	36.397
0.8070	37.063
0.9142	37.882
1.0807	39.022
1.4075	40.883

(b) C3 *adaptive*

Table 13. Raw values of our proposed method, C3, on the Kodak image benchmark.

Rate [bits per pixel]	PSNR [dB]
0.0613	29.190
0.0917	30.562
0.1045	31.030
0.1384	32.038
0.1588	32.540
0.1886	33.180
0.2382	34.103
0.3481	35.700
0.3923	36.241
0.4970	37.328
0.5582	37.855
0.6437	38.517
0.7814	39.453
1.0635	40.987

(a) C3

Rate [bits per pixel]	PSNR [dB]
0.0538	29.108
0.0852	30.524
0.0989	31.013
0.1335	32.040
0.1538	32.540
0.1844	33.180
0.2347	34.118
0.3451	35.713
0.3887	36.236
0.4946	37.326
0.5540	37.851
0.6406	38.523
0.7768	39.453
1.0594	40.991

(b) C3 *adaptive*

Table 14. Raw values of our proposed method, C3, on the CLIC2020 professional validation dataset split image benchmark.

Rate [bits per pixel]	PSNR [dB]
0.0159	32.926
0.0227	33.967
0.0331	35.027
0.0546	36.328
0.0846	37.450
0.1276	38.364
0.2540	39.705
0.4425	40.795

(a) C3 on all UVG videos

Rate [bits per pixel]	PSNR [dB]
0.0101	32.975
0.0124	33.410
0.0161	33.735
0.0242	34.065
0.0420	34.289
0.1082	34.936
0.4446	36.891
0.9688	38.808

(b) C3 on Beauty

Rate [bits per pixel]	PSNR [dB]
0.0103	35.567
0.0143	36.722
0.0203	37.919
0.0325	39.322
0.0530	40.383
0.0737	41.239
0.1382	42.429
0.2078	43.286

(c) C3 on Bosphorus

Rate [bits per pixel]	PSNR [dB]
$8.8842 \cdot 10^{-3}$	35.782
0.0107	36.516
0.0131	37.126
0.0179	37.793
0.0313	38.392
0.0414	38.736
0.0888	39.198
0.1987	39.889

(d) C3 on Honeybee

Rate [bits per pixel]	PSNR [dB]
0.0154	31.971
0.0217	33.017
0.0321	34.098
0.0551	35.462
0.0861	36.861
0.1289	37.866
0.2235	39.126
0.3799	40.033

(e) C3 on Jockey

Rate [bits per pixel]	PSNR [dB]
0.0344	29.710
0.0502	31.278
0.0732	32.884
0.1182	34.938
0.1685	36.799
0.2376	38.199
0.3640	40.092
0.5231	41.219

(f) C3 on Readysetgo

Rate [bits per pixel]	PSNR [dB]
0.0114	32.945
0.0168	33.901
0.0261	34.950
0.0451	36.195
0.0759	37.219
0.1107	37.956
0.2100	38.823
0.3895	39.846

(g) C3 on Shakendry

Rate [bits per pixel]	PSNR [dB]
0.0212	31.529
0.0328	32.923
0.0512	34.475
0.0892	36.520
0.1353	38.208
0.1930	39.616
0.3088	41.375
0.4301	42.480

(h) C3 on Yachtride

Table 15. Raw values of our proposed method, C3, UVG video benchmark; average rate and PSNR over the seven 1080p videos.