MobileNVC: Real-time 1080p Neural Video Compression on a Mobile Device

Ties van Rozendaal, Tushar Singhal, Hoang Le, Guillaume Sautiere, Amir Said,
Krishna Buska, Anjuman Raha, Dimitris Kalatzis, Hitarth Mehta, Frank Mayer, Liang Zhang,
Markus Nagel, Auke Wiggers

ties@qti.qualcomm.com, auke@qti.qualcomm.com

Qualcomm AI Research*

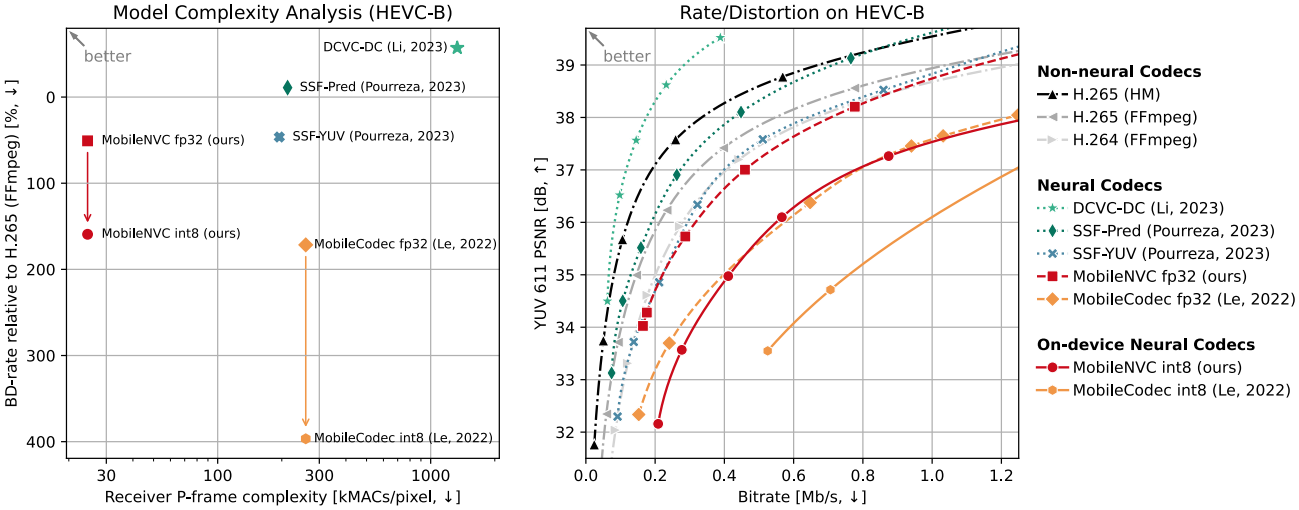

Figure 1. Compression performance versus receiver compute (left) and corresponding rate-distortion curves (right). Bjøntegaard Delta-rate is calculated with H.265 (FFmpeg, preset fast) as reference. The kMACS/pixel are computed using a full-HD 1080×1920 input.

Abstract

Neural video codecs have recently become competitive with standard codecs such as HEVC in the low-delay setting. However, most neural codecs are large floating-point networks that use pixel-dense warping operations for temporal modeling, making them too computationally expensive for deployment on mobile devices. Recent work has demonstrated that running a neural decoder in real time on mobile is feasible, but shows this only for 720p RGB video.

This work presents the first neural video codec that decodes 1080p YUV420 video in real time on a mobile device. Our codec relies on two major contributions. First, we design an efficient codec that uses a block-based motion compensation algorithm available on the warping core of the mobile accelerator, and we show how to quantize this model to integer precision. Second, we implement a fast decoder pipeline that concurrently runs neural network components on the neural signal processor, parallel entropy coding on the mobile GPU, and warping on the warping core. Our codec outperforms the previous on-device codec by a large margin with up to 48 % BD-rate savings, while reducing the MAC count on the receiver side by $10\times$. We perform a careful ablation to demonstrate the effect of the introduced motion compensation scheme, and ablate the effect of model quantization.

1. Introduction

Neural video compression has seen significant progress in recent years. In particular, in the *low-delay P* setting, various works [23, 24, 33] have outperformed reference implementations of standard codecs like HEVC (HM) [45] and VVC (VTM) [56] in compression performance. However, current neural codecs are computationally expensive

*Qualcomm AI Research is an initiative of Qualcomm Technologies, Inc.

compared to standard solutions, and reported runtimes are often measured on powerful desktop or datacenter GPUs. Additionally, many works assume the availability of pixel-based [1, 33, 35] or feature-based [18, 22–24] warping operations, which may be hard to efficiently implement on resource-constrained devices such as mobile phones.

Standard codecs, on the other hand, have fast software implementations such as FFmpeg [45, 59], or efficient silicon implementations specifically designed for fast decoding on consumer hardware. Although some works design neural codecs with efficiency in mind [9, 26, 35, 42, 54], the only published work that reports runtime on a mobile device is MobileCodec [21]. One of the main contributions of MobileCodec is to replace optical flow warping with a convolutional motion compensation network, avoiding the need to implement the warping operation on-device. However, this design has a negative impact on compression performance.

In this work, we build MobileNVC, a neural P-frame codec architecture designed for deployment to a mobile device. Instead of replacing warping, MobileNVC introduces a block-based warping scheme that can be implemented efficiently using a motion compensation kernel available on the Snapdragon® [1] 8 Gen 2 neural accelerator. Our network design is based on the model architecture of [33], made more efficient by using a lean flow extrapolator that predicts the next flow for blocks of pixels, and by using only few warping operations.

We improve inference efficiency by quantizing weights and activations to 8-bit integers. We show that naive quantization of the mean parameter of a mean-scale hyperprior leads to catastrophic loss in R-D performance, and propose a solution for low-precision quantization. For the scale parameter, we use the efficient quantization scheme of Said et al. [40]. We further increase throughput by implementing a parallel entropy coding algorithm on GPU, massively increasing parallelism to hundreds of threads [38], compared to the eight threads used by MobileCodec.

Together, these techniques enable extremely efficient neural video decoding on a mobile device. We obtain 48% Bjøntegaard Delta-rate savings compared to MobileCodec, and a $10\times$ reduction in computational complexity. Additionally, where MobileCodec only decodes HD (720p) video, we enable running $>$30fps full-HD (1080p) real-time decoding on mobile. We study the effect of the choice of warping operator and quantization in careful ablation studies, allowing us to determine key factors for effective mobile-friendly design of neural video codecs. Key results on compression performance and computational efficiency are shown in Figure 1.

[1] Snapdragon branded products are products of Qualcomm Technologies, Inc. and/or its subsidiaries.

2. Related work

2.1. Neural data compression

Neural codecs are systems that learn to compress data from examples. The most widely adopted model for neural data compression is the mean-scale hyperprior [5, 29]. This model is a hierarchical variational autoencoder with quantized latent variables, and can be seen as a specific version of a *compressive autoencoder* [50].

After initial success in the image domain [4, 29, 36], neural codecs were extended to the video setting [10, 12, 25]. Inspired by standard codecs, neural video codecs adopted motion compensation and residual coding using task-specific networks [1, 25, 37]. These architectures were further augmented using predictive models that predict the flow, residual or both [17, 33, 35], leading to improvements in compression performance. Recent works show that conditional coding can be more powerful than residual coding [23, 24], and perform similarly to the strongest standard video codecs. However, these architectures require multi-stage training to prevent aggregating error, and introduce various custom operations, such as feature-space motion compensation [18].

2.2. Efficient neural video codecs

Neural codecs are quickly closing the gap with standard codecs, but improved compression performance typically comes with an increase in computational cost [64]. For this reason, many works now report runtime and show the number of Multiply-Accumulate (MAC) operations. However, runtime is typically measured on desktop or datacenter GPUs. The deployment of neural codecs to resource-constrained devices has received relatively little attention.

In the learned image compression setting, early works improved rate-distortion performance by introducing bigger and better prior models [7, 11, 29, 60]. Follow-up work reduced computational cost by careful prior model design [14], or via transformer-based architectures, where much of the efficiency comes from the ability to parallelize computation across independent sub-tensors [26, 64]. Additionally, both Galpin et al. [9] and Yang et al. [63] show that highly asymmetric encoder-decoder architectures allow using a receiver with much lower MAC count, and EVC [57] shows that distillation and pruning can also prove effective.

In the learned video compression setting, various works aim to reduce the computational cost of the receiver [21, 35, 42, 54]. For instance, ELF-VC [35] designs a specific convolutional block to improve inference speed and BD-rate. AlphaVC introduces a technique that allows skipping tokens during entropy decoding, reducing runtime [42]. Van Rozendaal et al. [53, 54] show that by overfitting the codec to the instance to compress, one can drastically reduce computational cost on the receiver side.

Nevertheless, most neural video codecs include operations that are difficult to implement efficiently on-devices where the size of the memory is constrained. Examples include advanced motion compensation algorithms such as *scale-space warping* [1, 33, 35] or warping in feature-space using deformable convolutions [17, 18, 23, 24, 34]. To avoid these operations, some codecs replace motion compensation entirely, for example by only modeling the relation between frames via the prior model [27].

The main baseline for our work, MobileCodec [21], is the only work that demonstrates decoding of video on a mobile device in real-time. It achieves this by replacing the warping operation by a learned motion compensation subnetwork, quantizing the weights and activations, and by implementing parallel entropy coding on mobile CPU [39, 40].

2.3. Quantizing neural codecs

A common methodology for computational cost reduction is quantization of weights and activations. For neural *image* compression, one of the first works [3] studying neural quantization was mainly motivated by cross-platform reproducibility, as entropy coding is sensitive and may break due to non-deterministic floating-point operations. investigated Post-Training Quantization (PTQ) [13, 20, 41, 49] and Quantization-Aware Training (QAT) [16, 46–48] techniques for both weights and activations, with the aim to close the rate-distortion gap between the integer-quantized models and their floating-point counterparts. For instance, Sun et al. [49] introduce channel splitting, where the convolution output channels most sensitive to quantization are split up and quantized using a custom dynamic range, while other channels are pruned away. Various works [41, 46, 48, 49] have shown that using *per-channel* activation quantization can be effective. However, this requires bit shifts on the accumulator which is not commonly supported on most fixed-point accelerators. We therefore use the commonly supported, but less flexible, *per-tensor* quantization for the activations [30]. Note that all works above perform simulated quantization. Instead, we implement and benchmark the performance of the quantized model on a mobile device.

3. Method

We first describe the architecture, block-based warping scheme, and training losses needed to train a 32-bit floating point model. We then describe the quantization procedure, and how we run entropy coding and inference on-device.

3.1. Network architecture

The MobileNVC architecture is a variation of the scale-space flow [1] architecture of Agustsson et al. Its input and output heads are modified for YUV 4:2:0 inputs following Pourreza et al. [33]. The motivation for operating in the YUV color space is that distance in this space is better aligned with human perception [51], and the 4:2:0 subsampling scheme exploits the difference in sensitivity of the human eye between luminance and color.

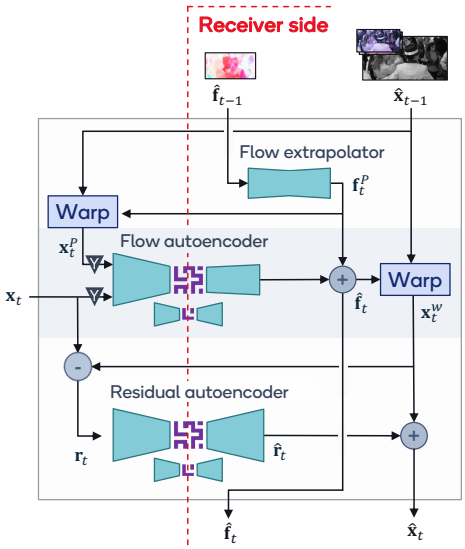

Figure 2. Architecture of our P-frame model. Due to the overlap block-based warping, the motion vectors are lower dimensional leading to reduced compute. The flow encoder is further optimized by only using the Y-channel of the inputs. [2]

The model consists of three mean-scale hyperprior [29] autoencoders and a flow extrapolator. The first mean-scale hyperprior acts as the *I-frame* model, compressing the first frame in a Group of Pictures (GoP) independent of other frames. The model for each subsequent *P-frame* consists of the remaining two autoencoders and the flow extrapolator, as visualized in Figure 2. Compressing a P-frame consists of three steps. First, the flow extrapolator predicts a flow $\mathbf{f}_t^P$ based on the previously transmitted flow $\hat{\mathbf{f}}_{t-1}$ (which we set to zero for frame x_1). Using this flow, we use warping (on the sender-side) to obtain an initial prediction for the next frame, $\mathbf{x}_t^P$. Second, we transmit a flow delta. The Y-channels of both $\mathbf{x}_t^P$ and the current ground truth frame $\mathbf{x}_t$ are given to the flow autoencoder, which estimates and compresses the flow residual $\mathbf{f}_t^P - \hat{\mathbf{f}}_t$. The reconstruction by the flow autoencoder is then added to the extrapolated flow to obtain the refined reconstructed flow $\hat{\mathbf{f}}_t$. We warp the previous predicted frame $\hat{\mathbf{x}}_{t-1}$ with $\hat{\mathbf{f}}_t$ to form the refined prediction $\mathbf{x}_t^w$. Third, the residual autoencoder compresses the frame residual $\mathbf{r} = \mathbf{x}_t - \mathbf{x}_t^w$. The resulting reconstruction $\hat{\mathbf{r}}$ is added to the warped frame on the receiver-side in order to form the final predicted frame $\hat{\mathbf{x}}_t$. Full details on network architecture can be found in Figure 6 in the Appendix.

[2] Image data from Tango video from Netflix Tango in Netflix El Fuente. Video produced by Netflix, with CC BY-NC-ND 4.0 license: https://media.xiph.org/video/derf/ElFuente/Netflix_Tango_Copyright.txt

3.2. Efficient block-based warping

Motion compensation is an essential component of both standard and neural video codecs. In this work, we use a block-based motion compensation scheme that has two main advantages over pixel-based schemes. First, it is possible to implement this scheme efficiently on the mobile neural accelerator. Second, by warping block-by-block, the flow tensor is of lower spatial dimensionality than the frame, reducing the computational cost of the flow autoencoder and extrapolator networks.

We first describe traditional, pixel-dense optical flow warping. A frame $\mathbf{x}$ is warped using a flow field $\mathbf{f}$, which is a 2D map indicating horizontal and vertical displacements. Specifically, for every pixel i, j in the warped frame, the value is retrieved from the reference frame as follows:

$$\text{warp}_{\text{dense}}\left(\mathbf{x}, \mathbf{f}\right)_{i,j} = \mathbf{x}[i + \mathbf{f}_x[i,j], j + \mathbf{f}_y[i,j]]. \quad (1)$$

Here $[\cdot]$ refers to array indexing, x and y sub-indices indicate retrieval of the respective coordinate in the vector field $\mathbf{f}$. For non-integer motion vectors, bilinear or bicubic interpolation is typically used to compute the pixel intensity.

The motion vector $\mathbf{f}$ often contains large homogeneous regions, as large objects and the background rarely show chaotic motion. Therefore, block-based warping can be used as a computationally efficient alternative to pixel-space warping. Here, the warped frame is divided into blocks of size $b \times b$, and all pixels in a block are retrieved from the reference frame using a single shared motion vector. The frame is thus warped as follows:

$$\text{warp}_{\text{block}}\left(\mathbf{x}, \mathbf{f}, b\right)_{i,j} = \mathbf{x}\left[i + \mathbf{f}_x\left[\left\lfloor\frac{i}{b}\right\rfloor, \left\lfloor\frac{j}{b}\right\rfloor\right], j + \mathbf{f}_y\left[\left\lfloor\frac{i}{b}\right\rfloor, \left\lfloor\frac{j}{b}\right\rfloor\right]\right]. \quad (2)$$

Block-based warping can be more efficient than dense pixel warping due to the block-wise memory access. However, one downside is that artifacts might occur around the block edges when adjacent blocks have different motion vectors. This can be solved using *overlapped block motion compensation*. Here, each block is warped multiple times using the $N-1$ surrounding motion vectors, and the results are averaged using a kernel $\mathbf{w} \in \mathbb{R}^{b \times b \times N}$ that decays towards the end of the blocks [31, 32], here a Gaussian:

$$\text{warp}_{\text{block-overlap}}\left(\mathbf{x}, \mathbf{f}, \mathbf{w}, b\right)_{i,j} = \sum_{k=1}^{N} w_{i,j,k} \cdot \mathbf{x}\Bigg[i + \mathbf{f}_x\left[\left\lfloor\frac{i}{b}\right\rfloor + b \cdot \Delta_x^k, \left\lfloor\frac{j}{b}\right\rfloor + b \cdot \Delta_y^k\right], j + \mathbf{f}_y\left[\left\lfloor\frac{i}{b}\right\rfloor + b \cdot \Delta_x^k, \left\lfloor\frac{j}{b}\right\rfloor + b \cdot \Delta_y^k\right]\Bigg], \quad (3)$$

where Δ^k defines the relative position of the neighboring block, e.g. $(-1,-1)$ for the top-left block, $(-1,0)$ for the center-left block.

We deploy an overlapped block motion compensation available in the mobile neural accelerator. As we will show in Section 5.3, this leads to better compression performance than block warping, matches that of dense pixel-space warping, and improves computational efficiency.

3.3. Loss functions

Models are trained using a loss consisting of a rate term, a distortion term, and auxiliary losses for the flow components. Similar to previous work, the rate loss is the sum of negative log-likelihoods of the latents and hyperlatents for the three auto-encoders [1]. Specific to our setup is that we use a zero-centered normal distribution with learned variance as the entropy model for the hyper-latent, instead of the non-parametric hyperprior from Ballé et al. [4]. This enables us to use the same entropy coding algorithm for latent and hyper-latent. We use rounding of latents and hyper-latents at evaluation time, but use a "mixed" quantization scheme during training: additive noise quantization when computing the rate loss, and rounding when computing the distortion losses [11]. We reweigh the mean squared error (MSE) distortion losses for the Y:U:V channels with weights 6:1:1, to align with the evaluation metrics [33, 44]:

$$D(\mathbf{x}, \hat{\mathbf{x}}) = \tfrac{6}{8}\,\text{MSE}_Y + \tfrac{1}{8}\,\text{MSE}_U + \tfrac{1}{8}\,\text{MSE}_V\,. \quad (4)$$

One challenge of training small models at lower bitrates is that frame quality deteriorates over time due to error accumulation. To account for this, we use an exponentially modulated P-frame loss that places emphasis on later frames, inspired by schemes that weigh the loss for each frame in the GoP differently [23, 35]:

$$D_{\text{mod}}(\mathbf{x}, \hat{\mathbf{x}}, \tau) = \frac{T}{\sum_{i=0}^{T-1} \tau^i} \sum_{i=0}^{T-1} \tau^i D(\mathbf{x}_i, \hat{\mathbf{x}}_i) \quad (5)$$

Additionally, we halve the value of the rate loss multiplier for I-frames, such that the PSNR value for the chosen operating point of I-frames and P-frames becomes more similar [22]. Lastly, we use auxiliary flow losses during training of our floating point model, to force the network to learn meaningful extrapolated and reconstructed flow fields. For both flow outputs $\mathbf{f} \in \{\mathbf{f}^p, \hat{\mathbf{f}}\}$, we set $D_{\text{flow}}(\mathbf{f}, \hat{\mathbf{x}}_{t-1}, \mathbf{x}_t) = D\left(\text{warp}\left(\hat{\mathbf{x}}_{t-1}, \mathbf{f}\right), \mathbf{x}_t\right)$. Our final loss is then a weighted combination of all loss terms:

$$\mathcal{L}(\mathbf{x}) = \beta R(\mathbf{x}_0) + D(\mathbf{x}_0, \hat{\mathbf{x}}_0) + 2\beta R(\mathbf{x}_{>0}) + D_{\text{mod}}(\mathbf{x}_{>0}, \hat{\mathbf{x}}_{>0}, \tau) + \lambda D_{\text{flow}}(\mathbf{f}^P) + \lambda D_{\text{flow}}(\hat{\mathbf{f}}). \quad (6)$$

We train one model for each value of β. We show the values of λ and τ for different training stages in Tab. 5 in the Appendix.

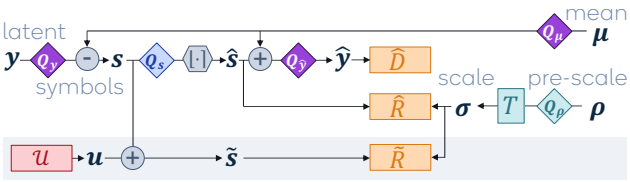

Figure 3. Computational graph of the latent bottleneck and entropy model during training. During floating point training, only the symbols $\mathbf{s}$ are quantized using rounding (hexagonal rounding operator), and a proxy rate loss $\tilde{R}$ based on additive quantization noise $\mathbf{u}$ is used (bottom pathway). During quantization-aware training, quantizers (shown as diamonds) are added to the graph. Quantizers with the same color and symbol have tied grids.

3.4. Integer Model Quantization

After training a model in 32-bit floating point, we quantize weights and activations to 8-bit integer precision using the AIMET library [43] in two stages. In the Post-Training Quantization (PTQ) stage, we estimate the quantizer parameters by passing a small amount of data through the network using a per-layer MSE loss [30]. To improve performance, we then add a Quantization-Aware Training (QAT) stage, where we use LSQ [8] to finetune both the network and quantization parameters using gradient descent. The exact hyperparameters of each stage can be found in Tab. 5.

We use integer quantization with a learned uniform grid, defined by a step size and a zero offset parameter. For the network weights, we learn a grid per output channel without zero offset, i.e., symmetric per-channel quantization. For activations, we learn a single quantization grid with a scale and zero offset, i.e., asymmetric *per-tensor* quantization.

A few operations require custom quantization grids. As noted in recent works [20, 41], bottleneck quantization in a mean-scale hyperprior architecture needs careful consideration when quantizing activations and performing entropy coding. The computational graph of the entropy model for the bottleneck of a mean-scale hyperprior is shown in Figure 3. We will first describe this graph without considering model quantization and then describe how we set the quantizers (shown as diamonds in this plot).

The latents $\mathbf{y}$ are the output of the mean-scale hyperprior encoder. These are not transmitted directly, rather, we transmit the *symbols* $\mathbf{s} = \mathbf{y} - \boldsymbol{\mu}$, which are the latents after mean subtraction. During inference, symbols are rounded to the integer grid $\hat{\mathbf{s}} = \lfloor \mathbf{s} \rceil$. During training, rounding is simulated in a differentiable manner using uniformly sampled additive noise $\mathbf{u} \sim U[-0.5, 0.5]$, resulting in $\tilde{\mathbf{s}} = \mathbf{s} + \mathbf{u}$ (Figure 3, bottom path in grey). A proxy rate loss $\tilde{R}$ is then based on these noisy latents, while the distortion loss $\hat{D}$ is based on the quantized latents [11].

To quantize the model for on device inference, we add quantizers for each of the variables, indicated by diamonds in Figure 3. The work of Said et al. [40] shows how to best quantize the scale σ: let the network predict a *pre-scale* $\rho \in (0, 1]$, which is mapped to the scale using an exponential-polynomial function $\sigma = T(\rho)$ by the entropy decoder. As the domain of this function is fixed, we can fix the quantizer Q_ρ to the same grid. At inference, ρ is directly passed to the entropy coding algorithm in int8.

The remaining question is how to choose quantizers for the latents, mean and symbols. As symbols are rounded to the integer grid, we choose their quantizer Q_s to have a symmetric grid with step size 1. For the pre-quantized latents $\mathbf{y}$ and the mean $\boldsymbol{\mu}$, we show experimentally that sub-integer precision is required for good rate-distortion performance, both in the PTQ and QAT setting. Specifically, we show that this problem can be solved by either using an 8-bit quantizer with a step size $\frac{1}{5}$ ($\frac{1}{3}$ for the highest four bitrates) or by using a 16-bit quantizer for $\mathbf{y}, \hat{\mathbf{y}}$, and $\boldsymbol{\mu}$. We observed that careful alignment of the grids performs better than learning a quantization grid via backpropagation. We emphasize that during the QAT stage, the rate loss is based on latents perturbed with uniform quantization noise.

3.5. Entropy Coding and Pipelined Inference

Our decoder must reach a throughput of 30+ frames per second (FPS) for full HD (1080 x 1920 pixels) YUV 4:2:0 videos. To best utilize available compute, we design an inference pipeline that uses different subsystems of the mobile neural processing unit (NPU). This pipeline is shown in Figure 4. Data from up to three timesteps are processed in parallel by simultaneously using the GPU, NPU, CPU and the warping kernel. Additionally, where MobileCodec used the CPU to perform entropy coding [21], our GPU implementation can easily be set to run entropy coding exclusively, ensuring stable performance and framerate.

As the mobile NPU and GPU can share memory, there are no delays caused by copying data between processing elements. The amount of entropy coding data to be processed is minimized by using only 8-bit integers for data elements. Arithmetic coding functions are implemented with OpenCL, and thanks to the small size of their tables, parallelization is defined mostly by the number of OpenCL work items, and work group organization is not critical [19].

4. Experiments

Training stages Training consists of two floating point training stages and two quantization stages. The hyperparameters for these training stages can be found in Tab. 5 in the Appendix. The first two stages train the floating-point model using the loss of Equation 6. In stage one, auxiliary losses for the flow are used, and stage two removes these by setting the loss factor $\lambda = 0$. In stage three, we perform PTQ by fitting the quantizers while keeping the model parameters fixed. In stage four, we perform QAT, by finetuning the model and quantizer parameters using LSQ [8].

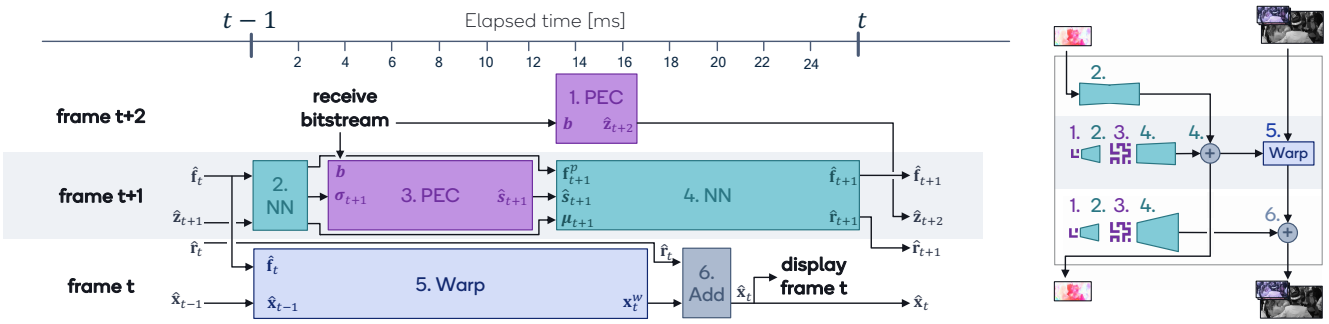

Figure 4. Pipelining of decoding stages. We show each stage and its inputs and outputs. Colors indicate on which component of the chip the stage runs: Neural networks (NN) on the neural accelerator, parallel entropy decoding (PEC) on the GPU, overlap block warping (Warp) on the warping kernel, and addition (ADD) on the CPU. Approximate runtime is indicated by the block width.

Datasets We train all models on Vimeo90k [62]. We use the Xiph-5N dataset [55, 61] as validation set, and tune hyperparameters on this set. We evaluate on multiple video compression benchmarks: HEVC-B test sequences [15], UVG-1k [28] sequences, and MCL-JVC [58] sequences.

Neural Baselines We compare our models against state-of-the-art neural video compression methods that report YUV performance like SSF-YUV and SSF-Pred from Pourreza et al. [33] and DCVC-DC from Li et al. [24]. The latter does not report performance on full video sequences, and we re-evaluate their multi-rate model, adjusting the quantization scales for the I-frame and context model to obtain performance at lower bitrates.

The main neural baseline is MobileCodec [21], as it is the only work to report results from a mobile device implementation. This model was not designed for YUV color space. We therefore train it on RGB for 1M steps and then finetune it on YUV 6:1:1 R-D loss for another 500k steps. We use the best available group of pictures (GoP) size for every model. This is GoP=32 for Li et al., GoP=∞ for Pourreza et al., and GoP=16 for our model and MobileCodec.

Standard Baselines We run H.265 and H.264 using FFmpeg [52] with the 'fast' preset. We also run HM [45] with default configurations. Note that on-device implementations of this standard codec are typically less performant than this reference implementation. We disable B-frames and use a GoP=∞. We do not include VVC or AV1. Details and exact commands can be found in Appendix A.1.

Metrics To evaluate compression performance, we compute peak signal-to-noise ratio (PSNR) on the Y, U and V channels separately. In line with common evaluation protocols [44] we average PSNR over the Y:U:V channels with weights of 6:1:1. We use Bjøntegaard-Delta bitrate [6] (BD-rate) to summarize rate-distorion performance in a single metric, based on bits-per-pixel (bpp) and YUV 6:1:1 PSNR.

We discard rate-distortion points where the bitrate is unreasonably high (above 1.3 Mb/s). To ensure a fair comparison, we omit outlier points: we keep points only if all other methods have a point that falls in the same PSNR range.

We compute the MAC count for our models and neural baselines using DeepSpeed [2]. Details can be found in Table 6 in the Appendix. For the MobileCodec-int8 model [21] we report AIMET [43] simulated rate-distortion results. To evaluate MobileNVC, we do not rely on simulated performance, but rather evaluate the actual rate-distortion resulting from on-device encoding and decoding.

5. Results and ablations

5.1. Rate-distortion and model complexity

Figure 1 shows BD-rate versus receiver complexity (left), as well as the rate-distortion performance curves (right). We summarize BD-rate results in Tab. 1. The best-performing floating-point neural codec, DCVC-DC, also has the largest model complexity (about $50\times$ more MAC operations than our MobileNVC model). Our floating-point model (MobileNVC fp32) has the lowest receiver-side complexity at 24.5 kMACs/pixel, and matches the BD-rate of the SSF-YUV model, despite having an $\sim$ 8 times lower MAC count. Our floating point codec still underperforms H.265 (FFmpeg). However, compared to MobileCodec, which was also designed for on-device inference, we improve compression performance by 45 % whilst reducing model complexity by more than $10\times$.

Our quantized codec (MobileNVC int8) far outperforms the quantized MobileCodec int8 – the only other work to show real-time mobile video decoding – with 48 % BD-rate savings. The performance gap between quantized models and state-of-the-art floating point codecs is substantial, but this is not surprising, as being able to decode on a mobile device poses tight computational constraints.

As we optimized for receiver inference speed, our receiver has 25% of the kMACs of our sender, whereas for

	BD rate metric	YUV:611 PSNR		Y PSNR	
	Dataset	HEVC-B	MCL	HEVC-B	MCL
int8	MobileNVC	**159.2**	**192.6**	**124.9**	**165.2**
	MobileCodec [21]	396.7	549.0	319.6	570.7
float32	MobileNVC	50.8	56.4	32.5	41.9
	MobileCodec [21]	171.6	294.4	145.2	268.8
	SSF-YUV [33]	46.4	44.2	25.7	27.2
	SSF-Pred [33]	-10.9	-0.1	-24.1	-10.3
	DCVC-DC [24]	**-57.1**	-	-	-

Table 1. BD-rate saving (in %) relative to ffmpeg x.265 for HEVC-B and MCL-JCV datasets, lower is better. DCVC-DC is excluded for datasets where PSNR did not overlap with other methods. More benchmarks can be found in Figure 7 in the Appendix.

other models this is 70-80%. For our model, components dealing with motion are lower complexity than baselines due to the low-dimensional flow vectors, and the fact that motion-autoencoder only uses Y-channels. Warping operations are not taken into account in the MAC count, but as we show in Figure 4, it can be executed in parallel with neural inference, without runtime overhead. More details on model complexity can be found in Tab. 6 in the Appendix.

5.2. Inference Speed

We measure inference speed on a mobile phone with a Snapdragon 8 Gen 2 system-on-chip. On HEVC-B, we achieve an average receiver inference speed of 38.9 FPS. As our focus is receiver-side complexity, we did not optimize our transmitter pipeline and run all steps sequentially, resulting in an encoding rate of around 3 FPS. For decoding, Figure 4 shows the approximate duration of each step. Inference speed is bounded by network inference, and due to parallelization, the warping operation is not causing any overhead. Tab. 2 shows the speed of our parallel entropy coding implementation. The GPU allows us to greatly optimize coding speed by using a large number of threads, at the cost of using more bits due to the larger header. We choose to use 512 threads and implement the header naïvely, noting that an optimized implementation could reduce the bitrate overhead by 2x. All in all, our method not only improves the compression performance compared with MobileCodec but also the throughput, allowing us to operate on full-HD (1080×1920) instead of HD (720×1280) resolution.

5.3. Model Ablations

We ablate model design choices in Tab. 3. All models in this table are unquantized and are trained only for 1M steps.

First, we look into warping. The model with dense warping (row III) has better R-D performance than our overlapped block-warp model (row I). However, the gap is small, with 6 % BD-rate cost, and the dense warp model has more than 4x more MACs due to the higher flow dimensionality. Comparing *overlapped* block-warp to *vanilla* block warp (II) without overlap, we see the effectiveness of overlapping, which brings about 19 % BD-rate savings. Alternatively, one could use a flow-agnostic model as is done in MobileCodec [21]. In row IV, we include a variant of our network that uses a conditional convolutional network that can model warping implicitly, as shown in Figure 5 in the Appendix. With more than 50 % BD-rate increase compared to overlapped block-warp, it is suboptimal both in terms of compute and compression performance, showing the importance of warping. An example warped frame for each of the methods can be seen in Table 7 in the Appendix.

Next, we look into the probability model for the prior. We train a version of our model with a scale-only prior (V) instead of a mean-scale prior (I), as in MobileCodec. Compression performance is significantly reduced, with a 9.6 % increase in BD-rate, while efficiency gains are minimal.

Lastly, we quantify the effect of our second training stage, which increases GoP size and uses P-frame loss modulation as described in section 3.3. Row VI shows that finetuning the main model (row I) for 250K steps with this scheme results in 14 % BD-rate savings.

			Bitrate overhead [↓]	
# Threads	Device	Decoding time [↓]	Naïve	Optimized
1	CPU	20 ms	0.0 %	0.0 %
8	CPU	16 ms	0.6 %	0.6 %
256	GPU	18 ms	2.1 %	1.1 %
512	**GPU**	**11 ms**	**3.9 %**	2.0 %
1024	GPU	6 ms	7.0 %	3.6 %

Table 2. Inference speed and rate overhead for different parallelization strategies for entropy coding of the latents. The row in bold indicates the settings we used in our work.

5.4. Quantization ablation

Figure 1 shows that moving from floating point to int8 substantially reduces compression performance. We break down this reduction in Tab. 4. Row I shows the effect of quantizing symbols $\mathbf{s}$ and flow vectors $\mathbf{f}^P, \hat{\mathbf{f}}$. This leads to a 11.9 % increase in BD-rate (i.e., worse compression performance), mainly due to flow quantization, and provides an upper bound for quantization performance. In row II, we also quantize the weights using per-channel quantization grids, leading to a 14.2 % overall increase. When we also quantize all activations except for those in the latent bottleneck (row III), BD-rate increases to 54.2 %.

Due to interaction between rounding of the latent symbols (or adding uniform noise) and activation quantization, care should be taken when quantizing the latent bottleneck. Row IV shows that the scale-quantization of [40] allows us to quantize the scale to 8-bit with no loss in performance. As described in section 3.4, the symbols $\mathbf{s}$ are rounded to

			Model architecture		Params [M, ↓]		kMACs/px [↓]		
		Model	warping	prior	send	recv	send	recv	BD-rate [↓]
Baseline	I.	MobileNVC, no finetuning	overlap block	mean-scale	12.42	6.30	64.93	24.52	0.0 %
Warping	II.	block warp	block warp	mean-scale	12.63	6.30	64.93	24.52	19.2 %
	III.	dense warp	dense warp	mean-scale	12.50	6.23	153.67	113.59	-6.0 %
	IV.	conditional conv	no warp	mean-scale	9.77	6.12	80.72	57.35	51.2 %
Prior	V.	scale-only prior	overlap block	scale	11.86	5.74	63.84	23.44	9.6 %
Training	VI.	MobileNVC (+ finetuning)	overlap block	mean-scale	12.42	6.30	64.93	24.52	-14.0 %

Table 3. Model architecture ablation. All models are floating-point and have been trained for 1M steps, except for model VI, which trains with stages 1 and 2 (see Tab. 5 for details). Parameters and kMACS/px are shown for the P-frame model only, and are computed for a 1080×1920 YUV420 input frame. We refer to the Appendix for corresponding R-D-curves (Figure 8, Left).

	Quantization Strategy	Quantizer setup							BD-rate [↓]	
		W	A	Q_s	Q_μ	Q_y	Q_σ	$Q_{\hat{y}}$	PTQ	QAT
I.	Symbols and flow only	✗	✗	✓	✗	✗	✗	✗	11.9 %	-
II.	+ weights	✓	✗	✓	✗	✗	✗	✗	14.2 %	-
III.	+ activations	✓	✓	✓	✗	✗	✗	✗	54.2 %	-
IV.	+ scale	✓	✓	✓	✓	✗	✗	✗	54.2 %	-
V.	Fully quantized (latent step size=1)	✓	✓	✓	✓	— ✓step size 1 —			243.3 %	-
VI.	Fully quantized (latent step size=$\frac{1}{5}$ / $\frac{1}{3}$)	✓	✓	✓	✓	- ✓step size $\frac{1}{5}$ / $\frac{1}{3}$ -			101.9 %	42.5 %
VII.	Fully quantized (latents int16)	✓	✓	✓	✓	—— ✓int16 ——			54.2 %	25.6 %

Table 4. Quantization ablation. Values with ✗ are unquantized and ✓indicates values are quantized to int8. BD-rate is computed with respect to the floating point baseline on the Xiph-5N dataset (lower is better). R-D curves can be found in Figure 8 (Right) in the Appendix.

the integer grid, so we use a step size of 1 for their quantization. Using the same integer grid for the latents $\mathbf{y}$ and the mean $\boldsymbol{\mu}$ as for the symbols causes a dramatic drop in compression performance (row V), showing the sensitivity of the mean-scale hyperprior to quantization.

One way to overcome this large quantization gap is to use high precision quantization (int16) for the latents and the mean (row VII), but this would increase on-device runtime. We show in row VI that a carefully chosen quantization grid, with a quantization step size of $\frac{1}{5}$ for the latents and mean parameter, and a step size of $\frac{1}{3}$ for the highest three bitrates, results in a big performance improvement as well. This choice allows us to cover a sufficiently large range of values, and avoids that points on the sub-integer grid fall exactly between two points on the coarser grid, thereby reducing "tie-break" issues compared to for example using step size $\frac{1}{4}$. As this choice only requires int8 activations, it does not result in a runtime increase.

Lastly, we see in the rightmost column that compression performance after post-training quantization (PTQ) can be further improved using quantization-aware training (QAT). Row VI gives us our final BD-rate overhead of 42.5 % relative to the floating-point model. Row VII shows that if the runtime increase were acceptable, mixed precision would be the better choice from a compression performance perspective. For additional analysis, we refer the reader the rate-distortion curves in Figure 8 (right) in the Appendix.

6. Conclusion

In this work, we introduce a practical neural codec that performs real-time decoding of full HD video on a mobile device. This codec (MobileNVC), outperforms the previous state-of-the-art practical codec by 48% BD-rate, while reducing the Multiply-Accumulate count by $10\times$. We design an efficient network architecture using a new block-based motion compensation algorithm, and show how to pipeline inference to enable real time decoding on device. Careful ablations show the effect of the introduced motion compensation and quantization schemes.

Most neural codecs are still too computationally expensive to be adopted in real life settings. We therefore hope that this work advances the field of practical neural codecs, and that it encourages future authors to benchmark their compression algorithms on-device.

Acknowledgements We thank Reza Pourreza for help with training and adapting the SSFPred model. Thanks to Alireza Shoa, Asma Qureshi and Darren Gnanapragasam for advice on the warping kernel.

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 conference on Computer Vision and Pattern Recognition*, 2020. 2, 3, 4, 14

[2] Reza Yazdani Aminabadi, Samyam Rajbhandari, Minjia Zhang, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Jeff Rasley, Shaden Smith, Olatunji Ruwase, et al. Deepspeed inference: Enabling efficient inference of transformer models at unprecedented scale. *arXiv preprint arXiv:2207.00032*, 2022. 6

[3] Johannes Ballé, Nick Johnston, and David Minnen. Integer networks for data compression with latent-variable models. In *International Conference on Learning Representations (ICLR)*, 2018. 3

[4] 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*, 2018. 2, 4

[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*, 2018. 2

[6] Gisle Bjøntegaard. Calculation of average psnr differences between rd-curves. 2001. 6

[7] Zhengxue Cheng, Heming Sun, Masaru Takeuchi, and Jiro Katto. Learned image compression with discretized gaussian mixture likelihoods and attention modules. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7939–7948, 2020. 2

[8] Steven K. Esser, Jeffrey L. McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S. Modha. Learned step size quantization. In *International Conference on Learning Representations (ICLR)*, 2020. 5

[9] Franck Galpin, M. Balcilar, Frédéric Lefèbvre, Fabien Racap'e, and Pierre Hellier. Entropy coding improvement for low-complexity compressive auto-encoders. 2023. 2

[10] 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. 2

[11] Zongyu Guo, Zhizheng Zhang, Runsen Feng, and Zhibo Chen. Soft then hard: Rethinking the quantization in neural image compression. In *International Conference on Machine Learning*, pages 3920–3929. PMLR, 2021. 2, 4, 5

[12] Amirhossein Habibian, Ties van Rozendaal, Jakub M Tomczak, and Taco S Cohen. Video compression with rate-distortion autoencoders. In *IEEE International Conference on Computer Vision*, 2019. 2

[13] Dailan He, Ziming Yang, Yuan Chen, Qi Zhang, Hongwei Qin, and Yan Wang. Post-training quantization for cross-platform learned image compression, 2022. 3

[14] Dailan He, Ziming Yang, Weikun Peng, Rui Ma, Hongwei Qin, and Yan Wang. Elic: Efficient learned image compression with unevenly grouped space-channel contextual adaptive coding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5718–5727, 2022. 2

[15] HEVC. Common test conditions and software reference configurations. http://phenix.it-sudparis.eu/jct/doc_end_user/current_document.php?id=7281, 2013. 6

[16] Weixin Hong, Tong Chen, Ming Lu, Shiliang Pu, and Zhan Ma. Efficient neural image decoding via fixed-point inference. *IEEE Transactions on Circuits and Systems for Video Technology*, 31(9):3618–3630, 2021. 3

[17] Zhihao Hu, Guo Lu, Jinyang Guo, Shan Liu, Wei Jiang, and Dong Xu. Coarse-to-fine deep video coding with hyperprior-guided mode prediction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5921–5930, 2022. 2, 3

[18] Zhihao Hu, Guo Lu, and Dong Xu. FVC: A new framework towards deep video compression in feature space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1502–1511, 2021. 2, 3

[19] David Kaeli, Perhaad Mistry, Dana Schaa, and Dong Ping Zhang. *Heterogeneous Computing with OpenCL 2.0*. Morgan Kaufmann Publishers, Waltham, MA, 2015. 5

[20] Esin Koyuncu, Timofey Solovyev, Elena Alshina, and André Kaup. Device interoperability for learned image compression with weights and activations quantization. In *2022 Picture Coding Symposium (PCS)*, pages 151–155, 2022. 3, 5

[21] Hoang Le, Liang Zhang, Amir Said, Guillaume Sautière, Yang Yang, Pranav Shrestha, Fei Yin, Reza Pourreza, and Auke Wiggers. Mobilecodec: neural inter-frame video compression on mobile devices. In *MMSys*, pages 324–330. ACM, 2022. 2, 3, 5, 6, 7, 14

[22] Jiahao Li, Bin Li, and Yan Lu. Deep contextual video compression, 2021. 2, 4

[23] Jiahao Li, Bin Li, and Yan Lu. Hybrid spatial-temporal entropy modelling for neural video compression. In *Proceedings of the 30th ACM International Conference on Multimedia*, pages 1503–1511, 2022. 1, 2, 3, 4

[24] Jiahao Li, Bin Li, and Yan Lu. Neural video compression with diverse contexts. *arXiv preprint arXiv:2302.14402*, 2023. 1, 2, 3, 6

[25] Guo Lu, Wanli Ouyang, Dong Xu, Xiaoyun Zhang, Chunlei Cai, and Zhiyong Gao. DVC: An end-to-end deep video compression framework. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2019. 2

[26] Fabian Mentzer, Eirikur Agustsson, and Michael Tschannen. M2t: Masking transformers twice for faster decoding, 2023. 2

[27] Fabian Mentzer, George Toderici, David Minnen, Sergi Caelles, Sung Jin Hwang, Mario Lucic, and Eirikur Agustsson. VCT: A video compression transformer. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. 3

[28] Alexandre Mercat, Marko Viitanen, and Jarno Vanne. UVG dataset: 50/120fps 4k sequences for video codec analysis and development. In *ACM Multimedia Systems Conference*, pages 297–302, 2020. 6

[29] David Minnen, Johannes Ballé, and George D Toderici. Joint autoregressive and hierarchical priors for learned image compression. *Advances in neural information processing systems*, 31, 2018. 2, 3

[30] Markus Nagel, Marios Fournarakis, Rana Ali Amjad, Yelysei Bondarenko, Mart van Baalen, and Tijmen Blankevoort. A white paper on neural network quantization. *arXiv preprint arXiv:2106.08295*, 2021. 3, 5

[31] Satoshi Nogaki and Mutsumi Ohta. An overlapped block motion compensation for high quality motion picture coding. In *[Proceedings] 1992 IEEE International Symposium on Circuits and Systems*, volume 1, pages 184–187. IEEE, 1992. 4

[32] Michael T Orchard and Gary J Sullivan. Overlapped block motion compensation: An estimation-theoretic approach. *IEEE Transactions on Image Processing*, 3(5):693–699, 1994. 4

[33] Reza Pourreza, Hoang Le, Amir Said, Guillaume Sautière, and Auke Wiggers. Boosting neural video codecs by exploiting hierarchical redundancy. *CoRR*, abs/2208.04303, 2022. 1, 2, 3, 4, 6, 14

[34] Linfeng Qi, Jiahao Li, Bin Li, Houqiang Li, and Yan Lu. Motion information propagation for neural video compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6111–6120, 2023. 3

[35] Oren Rippel, Alexander G Anderson, Kedar Tatwawadi, Sanjay Nair, Craig Lytle, and Lubomir Bourdev. ELF-VC: Efficient Learned Flexible-Rate Video Coding. *Neural Information Processing Systems*, 2021. 2, 3, 4

[36] Oren Rippel and Lubomir Bourdev. Real-Time adaptive image compression. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 2922–2930. PMLR, 2017. 2

[37] Oren Rippel, Sanjay Nair, Carissa Lew, Steve Branson, Alexander G. Anderson, and Lubomir Bourdev. Learned video compression. In *IEEE International Conference on Computer Vision*, October 2019. 2

[38] Amir Said, Hoang Le, and Farzad Farhadzadeh. Bitstream organization for parallel entropy coding on neural network-based video codecs. In *IEEE Int. Symp. on Multimedia*, Dec. 2023. 2

[39] Amir Said, Abo-Talib Mahfoodh, and Sehoon Yea. Compressed data organization for high throughput parallel entropy coding. In *Applications of Digital Image Processing XXXVIII*, volume 9599, pages 528–536. SPIE, 2015. 3

[40] Amir Said, Reza Pourreza, and Hoang Le. Optimized learned entropy coding parameters for practical neural-based image and video compression. In *2022 IEEE International Conference on Image Processing (ICIP)*, pages 661–665. IEEE, 2022. 2, 3, 5, 7

[41] Junqi Shi, Ming-Tse Lu, and Zhan Ma. Rate-Distortion Optimized Post-Training Quantization for Learned Image Compression. 2022. 3, 5

[42] Yibo Shi, Yunying Ge, Jing Wang, and Jue Mao. AlphaVC: High-Performance and Efficient Learned Video Compression, 2022. 2

[43] Sangeetha Siddegowda, Marios Fournarakis, Markus Nagel, Tijmen Blankevoort, Chirag Patel, and Abhijit Khobare. Neural network quantization with ai model efficiency toolkit (aimet). *arXiv preprint arXiv:2201.08442*, 2022. 5, 6

[44] J Ström, K Andersson, R Sjöberg, A Segall, F Bossen, G Sullivan, JR Ohm, and A Tourapis. Working practices using objective metrics for evaluation of video coding efficiency experiments. *ITU-T and ISO/IEC, JTC*, 1:23002–8, 2020. 4, 6

[45] Gary J. Sullivan, Jens-Rainer Ohm, Woo-Jin Han, and Thomas Wiegand. Overview of the high efficiency video coding (hevc) standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 22(12):1649–1668, 2012. 1, 2, 6

[46] Heming Sun, Zhengxue Cheng, Masaru Takeuchi, and Jiro Katto. End-to-end learned image compression with fixed point weight quantization. In *2020 IEEE International Conference on Image Processing (ICIP)*, pages 3359–3363, 2020. 3

[47] Heming Sun, Lu Yu, and Jiro Katto. End-to-end learned image compression with quantized weights and activations, 2021. 3

[48] Heming Sun, Lu Yu, and Jiro Katto. Learned image compression with fixed-point arithmetic. In *2021 Picture Coding Symposium (PCS)*, pages 1–5, 2021. 3

[49] Heming Sun, Lu Yu, and Jiro Katto. Q-lic: Quantizing learned image compression with channel splitting. *IEEE Transactions on Circuits and Systems for Video Technology*, page 1–1, 2022. 3

[50] Lucas Theis, Wenzhe Shi, Andrew Cunningham, and Ferenc Huszár. Lossy image compression with compressive autoencoders. *International Conference on Learning Representations*, 2017. 2

[51] M Tkalcic and J F Tasic. Colour spaces: perceptual, historical and applicational background. In *The IEEE Region 8 EUROCON 2003. Computer as a Tool.*, volume 1, pages 304–308 vol.1, Sept. 2003. 3

[52] Suramya Tomar. Converting video formats with ffmpeg. *Linux Journal*, 2006(146):10, 2006. 6

[53] Ties van Rozendaal, Johann Brehmer, Yunfan Zhang, Reza Pourreza, and Taco S Cohen. Instance-adaptive video compression: Improving neural codecs by training on the test set. *arXiv preprint arXiv:2111.10302*, 2021. 2

[54] Ties van Rozendaal, Johann Brehmer, Yunfan Zhang, Reza Pourreza, Auke J. Wiggers, and Taco Cohen. Instance-adaptive video compression: Improving neural codecs by training on the test set. *Transactions on Machine Learning Research*, 2023. Expert Certification. 2

[55] Ties van Rozendaal, Iris AM Huijben, and Taco S Cohen. Overfitting for fun and profit: Instance-adaptive data compression. In *International Conference on Learning Representations (ICLR)*, 2021. 6

[56] VTM. Vtm reference software for versatile video coding (vvc). https://vcgit.hhi.fraunhofer.de/jvet/VVCSoftware_VTM/, 2013. 1

[57] Guo-Hua Wang, Jiahao Li, Bin Li, and Yan Lu. Evc: Towards real-time neural image compression with mask decay. *arXiv preprint arXiv:2302.05071*, 2023. 2

[58] 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. 6

[59] T. Wiegand, G.J. Sullivan, G. Bjontegaard, and A. Luthra. Overview of the h.264/avc video coding standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 2003. 2

[60] Lirong Wu, Kejie Huang, and Haibin Shen. A GAN-based tunable image compression system. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 2334–2342, 2020. 2

[61] Xiph.org. Video test media. https://media.xiph.org/video/derf/. 6

[62] Tianfan Xue, Baian Chen, Jiajun Wu, Donglai Wei, and William T Freeman. Video enhancement with task-oriented flow. *International Journal of Computer Vision (IJCV)*, 127(8):1106–1125, 2019. 6

[63] Yibo Yang and Stephan Mandt. Asymmetrically-powered neural image compression with shallow decoders. *arXiv preprint arXiv:2304.06244*, 2023. 2

[64] Yinhao Zhu, Yang Yang, and Taco Cohen. Transformer-based transform coding. In *International Conference on Learning Representations*, Mar 2022. 2