Boosting neural video codecs by exploiting hierarchical redundancy

Reza Pourreza, Hoang Le, Amir Said, Guillaume Sautière, Auke Wiggers
Qualcomm AI Research‡
{pourreza, hoanle, asaid, gsautie, auke}@qti.qualcomm.org

Abstract

In video compression, coding efficiency is improved by reusing pixels from previously decoded frames via motion and residual compensation. We define two levels of hierarchical redundancy in video frames: 1) first-order: redundancy in pixel space, i.e.*, similarities in pixel values across neighboring frames, which is effectively captured using motion and residual compensation, 2) second-order: redundancy in motion and residual maps due to smooth motion in natural videos. While most of the existing neural video coding literature addresses first-order redundancy, we tackle the problem of capturing second-order redundancy in neural video codecs via predictors. We introduce generic motion and residual predictors that learn to extrapolate from previously decoded data. These predictors are lightweight, and can be employed with most neural video codecs in order to improve their rate-distortion performance. Moreover, while RGB is the dominant colorspace in neural video coding literature, we introduce general modifications for neural video codecs to embrace the YUV420 colorspace and report YUV420 results. Our experiments show that using our predictors with a well-known neural video codec leads to 38% and 34% bitrate savings in RGB and YUV420 colorspaces measured on the UVG dataset.*

1. Introduction

Video traffic is predicted to reach more than 82% of all internet traffic in 2022 [6]. As video resolution and framerate demands have steadily increased over the years, there is a strong need to develop algorithms that enable transmission of video data at low bit cost. Standard video codecs such as AVC [34] and HEVC [27] were created exactly for this purpose.

The goal of a codec is to trade off the number of bits spent to transmit a message and the distortion between the original data and its reconstruction. In the video setting, standard codecs keep the bitrate low by using previously transmitted information as context. For example, to make a prediction for a current frame, the previously transmitted frame can be used. Instead of transmitting the current frame, the motion between the previous reconstruction and the current frame may be transmitted. This allows making an initial prediction for the current frame using motion compensation. Transmitting the motion typically has a lower bit cost than transmitting the frame, as motion of pixels belonging to the same object are strongly correlated. Similarly, an additive correction or *residual* can be transmitted to refine a receiver-side frame. If the residual is sparse, the cost of transmitting a residual will be much lower than transmitting the frame in isolation.

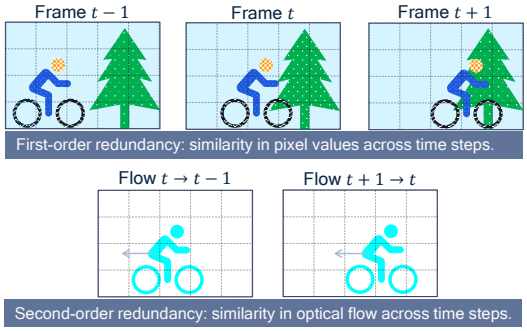

Figure 1: First- and second-order redundancy in video, here shown for (backward) optical flow. First-order redundancy exists between frames in pixel-space, second-order redundancy exists between optical flow maps.

Standard codecs take this idea one step further and make *motion vector predictions*. As motion vectors are typically similar across timesteps due to smooth object motion in video, an initial prediction can be made based on previous motion. This means the codec only needs to transmit corrections to this prediction, usually at low bit cost.

To distinguish these two approaches, we distinguish between two types of redundancy in videos. First-order redundancy is redundancy in the pixel domain. Inter-frame codecs exploit this using motion compensation and residual coding. Second-order redundancy is the redundancy in flow and residual maps due to smooth motions in videos, which can be exploited by making motion vector predic-

‡Qualcomm AI Research is an initiative of Qualcomm Technologies, Inc.
Preprint. Work in progress.

tions. These are shown schematically for motion in Fig. 1.

Most modern neural video codecs exploit only first-order redundancy using a two-step process [2, 11, 13, 17]: a *flow model* transmits motion information (optical flow) and *residual model* transmits residual maps. There are works that exploit second-order redundancy as well. For example, ELF-VC [23] predicts the flow, and Liu *et al.* [16] use flow and residual prediction in latent space.

In this work, we exploit second-order redundancy via pixel-space flow and residual predictors. We design generic models that predict flow and residual using the history available in a frame buffer. This leads to easy-to-interpret predictions compared to predictions in latent space, makes it possible to combine the predictors with many existing neural codecs, and delivers significant performance improvement.

Furthermore, we provide evaluations for multiple input colorspaces. Most literature on neural video codecs only provides solutions and evaluations for the RGB colorspace, with some exceptions [7, 24]. However, standard codecs were designed to operate with YUV420 input. YUV420 is a color format that subsamples the chroma channels of the YUV colorspace which is closer to how humans perceive images and leads to bandwidth saving in compression applications. To gauge progress in learned video compression compared to standard codecs, we train our codecs on both RGB and YUV420 input formats. We demonstrate that neural codecs can excel in both domains, and that only minor architectural modifications are needed to handle the difference in input resolution between colorspaces.

In summary, the contributions of this paper are:

1. We define the framework of hierarchical redundancy in video in order to categorize neural codecs, and observe that most neural codecs only address the first type of redundancy.
2. To exploit second-order redundancy, we introduce generic flow and residual predictors that can be paired with existing inter-frame codecs and deliver strong performance improvements.
3. We show that simple modifications enable codecs to run on YUV420 inputs, and report both RGB and YUV420 performance.

2. Related work

2.1. Learned compression

The focus of this work is a so-called *learned codec*, meaning each component is a learned model. Neural network-based codecs have been successfully applied to data compression in many domains, including the image [3, 28, 30] and video [2, 9, 11, 13, 17, 23] domain. In these systems, a neural encoder takes the data $\mathbf{x}$ and produces a quantized latent variable $\mathbf{z}$, and a neural decoder produces a reconstruction $\hat{\mathbf{x}}$ given this latent. A neural prior or context model is used to learn the distribution of latent variables $p(z)$. Using this prior and an entropy coding algorithm, Shannons source-coding theory tells us that the latent $\mathbf{z}$ can be losslessly compressed using $-\log p(\mathbf{z})$ bits.

Neural codecs are typically trained to minimize a rate-distortion loss consisting of two terms:

$$\mathcal{L}_{RD} = \mathbb{E}_{\mathbf{x}\sim p(x)} \left[\beta \mathcal{L}_{rate}(\mathbf{z}) + \mathcal{L}_{distortion}(\mathbf{x}, \hat{\mathbf{x}})\right]. \quad (1)$$

The rate term corresponds to the number of bits needed to transmit the quantized latent variable $\mathbf{z}$, and a distortion term corresponds to the distance between the reconstruction $\hat{\mathbf{x}}$ and the ground truth $\mathbf{x}$.

Neural codecs promise several advantages over handcrafted codecs. First, as they learn to identify redundancies from example data, they have shown a strong ability to specialize to a domain [9] or even a single datapoint [25, 31, 32]. For example, if the codec will only be used to code animated content, it can easily be fine-tuned to this domain. Second, neural codecs benefit from advances in general-purpose neural hardware, where most video codecs require dedicated hardware to run in real time on device. Third, they have shown to be able to hallucinate desirable textures, leading to improved perceptual quality compared to traditional codecs in user studies [18, 19, 38].

2.2. Neural video compression

In the video setting, neural codec design has been inspired by techniques from handcrafted codecs. Where early works used frame interpolation [35] or predicted entire blocks of frames jointly [9], more recent neural video codecs exploit similarity between frames using motion compensation and residual coding. Having neural networks learn to perform these steps end-to-end, as introduced by [17], has led to major bitrate savings in both the low latency [2, 11, 13, 23, 24], and streaming setting [14, 22].

As consecutive frames often contain continuous motion, both motion information and residuals are predictable given previous frames. Recent neural video codecs exploit this fact, for example, by adding components that predict the flow [23] or predict the motion and residual latents [16].

In this work, we use simple predictors for both flow and residual. By only incorporating context from the past two timesteps, and avoiding statefulness and recurrent components, we avoid instabilities due to aggregating error. Unlike ELF-VC [23], our codec requires no normalization layers anywhere in the network, which makes the codec easier to deploy to hardware [1, 21]. Unlike Liu [16], we make predictions in pixel-space. This makes predictions easy to interpret, and enables combining our predictors with different base models.

2.3. YUV420 colorspace

Most neural video codecs are trained to maximize PSNR or MS-SSIM in the RGB colorspace. However, when visual quality according to human observers is important, other colorspaces may be better suited. Distance in the YUV colorspace is known to match human perception more closely than distance in the euclidean RGB space [29].

Most standard codecs were designed to maximize PSNR in the YUV420 input domain. This is a subsampling of YUV, where chroma components are subsampled by $4\times$ to reduce bitrate and to meet bandwidth constraints. For a $H \times W$ input, this subsampling results in one $H \times W$ luma channel, and two $\frac{H}{2} \times \frac{W}{2}$ chroma channels.

There exist a few learned video codecs that support the YUV420 color format [7, 24], or have been trained to optimize metrics in the YUV domain [23]. However, as of yet, no learned video codec has shown competitive PSNR or MS-SSIM in the YUV420 domain.

3. Method

3.1. Base codec

In this work, we introduce flow and residual predictors to capture second-order redundancy in the low-delay neural video coding setting. Although the introduced predictors can potentially be added to any neural video codec, here, we build upon the well-known video codec scale-space flow (SSF) [2] due to the simple and efficient network design and the straightforward training schedule. The block-diagram of SSF in shown is Figure 2, where $\widehat{\mathbf{f}}_t$ indicates scale-space flow and Warp indicates scale-space warping.

Figure 4 (a) shows two consecutive video frames, and the associated optical flow and residual maps are shown in Figure 4 (b) and (c), respectively. Of course the flow and residual for the first timestep depend on a previous frame, which is omitted here for brevity. As can be seen from these figures, optical flow maps and residuals are much less detailed than video frames, making them easier to compress. Additionally, due to the smooth motion in this video, optical flow and residual maps are highly correlated across the two time steps.

3.2. Predictors

Here, we build a neural video codec on top of SSF where we keep the intra-frame codec as well the *flow-block* and *residual-block* in the inter-frame codec unchanged. We add flow and residual predictors to the base inter-frame codec as shown in Figure 3.

Inter-frame coding is done via a four-step process as follows: 1) *flow prediction*, 2) *flow compression*, 3) *residual prediction*, 4) *residual compression*. These four steps are explained below:

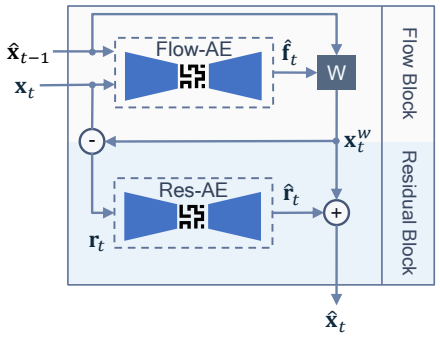

Figure 2: Base inter-frame codec architecture performing motion compensation and residual coding. W indicates a scale-space warp.

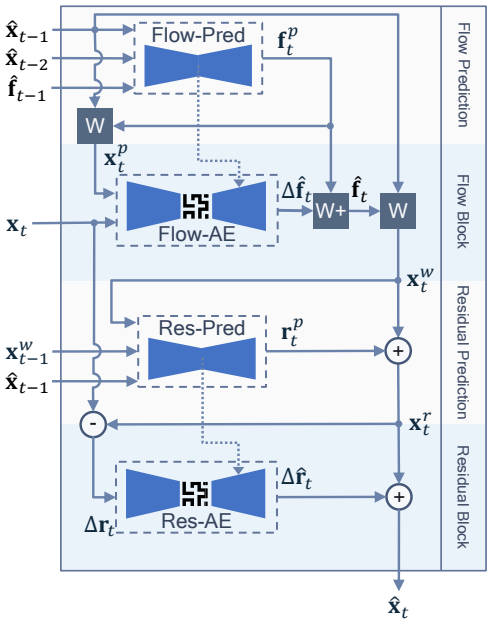

Figure 3: Architecture for our inter-frame model. W+ indicates a scale-space warp followed by a summation.

1. Flow prediction: as the name suggests, we predict the flow for the current time step $\mathbf{f}_t^p$ using a history of decoded frames, $\widehat{\mathbf{x}}_{t-1}$ and $\widehat{\mathbf{x}}_{t-2}$, and the previous flow $\widehat{\mathbf{f}}_{t-1}$, as follows:

$$\mathbf{f}_t^p = \text{Flow-Pred}(\widehat{\mathbf{x}}_{t-2}, \widehat{\mathbf{x}}_{t-1}, \widehat{\mathbf{f}}_{t-1}) \tag{2}$$

Here, Flow-Pred is a lightweight network that directly predicts $\mathbf{f}_t^p$ using the provided inputs. The predicted flow $\mathbf{f}_t^p$ is then used to generate a prediction of the current frame $\mathbf{x}_t^p$. This operation is done only on the transmitter side.

$$\mathbf{x}_t^p = \text{Warp}(\widehat{\mathbf{x}}_{t-1}, \mathbf{f}_t^p) \tag{3}$$

2. Flow compression: in this step, the predicted flow $\mathbf{f}_t^p$ is corrected using delta flow $\Delta\widehat{\mathbf{f}}_t$ that is measured and

transmitted via Flow-AE:

$$\Delta\hat{\mathbf{f}}_t = \text{Flow-AE}(\mathbf{x}_t^p, \mathbf{x}_t) \quad (4)$$

$$\hat{\mathbf{f}}_t = \Delta\hat{\mathbf{f}}_t + \text{Warp}(\mathbf{f}_t^p, \Delta\hat{\mathbf{f}}_t) \quad (5)$$

Here, Flow-AE is a hyperprior network identical to the one in SSF. The motion compensated frame $\mathbf{x}_t^w$ is generated using the corrected flow $\hat{\mathbf{f}}_t$:

$$\mathbf{x}_t^w = \text{Warp}(\hat{\mathbf{x}}_{t-1}, \hat{\mathbf{f}}_t) \quad (6)$$

3. Residual prediction: we predict the residual for the current time step $\mathbf{r}_t^p$ based on the motion compensated frames $\mathbf{x}_t^w$ and $\mathbf{x}_{t-1}^w$ as well as the previous decoded frame $\hat{\mathbf{x}}_{t-1}$:

$$\mathbf{r}_t^p = \text{Res-Pred}(\mathbf{x}_t^w, \mathbf{x}_{t-1}^w, \hat{\mathbf{x}}_{t-1}) \quad (7)$$

Here, Res-Pred is a lightweight network that directly predicts $\mathbf{r}_t^p$ using the provided inputs. Given that $\mathbf{r}_t = \mathbf{x}_t - \mathbf{x}_t^w$, the rationale is that Res-Pred should be able to predict $\mathbf{r}_t$ by looking at $\hat{\mathbf{x}}_{t-1}$ and $\mathbf{x}_{t-1}^w$. Next, $\mathbf{r}_t^p$ is applied to $\mathbf{x}_t^w$:

$$\mathbf{x}_t^r = \mathbf{x}_t^w + \mathbf{r}_t^p \quad (8)$$

4. Residual compression: in this step, the predicted residual $\mathbf{r}_t^p$ is corrected using delta residual $\Delta\hat{\mathbf{r}}_t$ that is transmitted via Res-AE::

$$\Delta\hat{\mathbf{r}}_t = \text{Res-AE}(\mathbf{x}_t - \mathbf{x}_t^r) \quad (9)$$

Here, Res-AE is a hyperprior network identical to the one in SSF. Finally, the decoded frame $\hat{\mathbf{x}}_t$ is generated as follows:

$$\hat{\mathbf{x}}_t = \mathbf{x}_t^r + \Delta\hat{\mathbf{r}}_t \quad (10)$$

The following points are worth emphasizing:

- All the used flows are scale-space flow and all the warp operations are scale-space warp [2].
- Both Flow-Pred and Res-Pred have hourglass architectures. The decoders of Flow-AE and Res-AE are conditioned on the bottleneck features of Flow-Pred and Res-Pred, respectively.
- We always encode the first frame in a sequence as an intra-frame and the subsequent frames as inter-frames. For the first inter-frame $\mathbf{x}_1$, note that $\hat{\mathbf{x}}_{t-2}$, $\hat{\mathbf{f}}_{t-1}$, and $\mathbf{x}_{t-1}^w$ are not available. They are set as follows:

$$\hat{\mathbf{x}}_{t-2} = \mathbf{x}_{t-1}^w = \hat{\mathbf{x}}_{t-1} \quad (11)$$

$$\hat{\mathbf{f}}_{t-1} = \mathbf{0} \quad (12)$$

- To construct the total flow $\Delta\hat{\mathbf{f}}_t$ in step 2, we choose to warp the predicted flow, which we observed to lead to better performance than additive operations. Other schemes have been used as well, such as letting by the Flow-AE output a scale-and-shift [23].

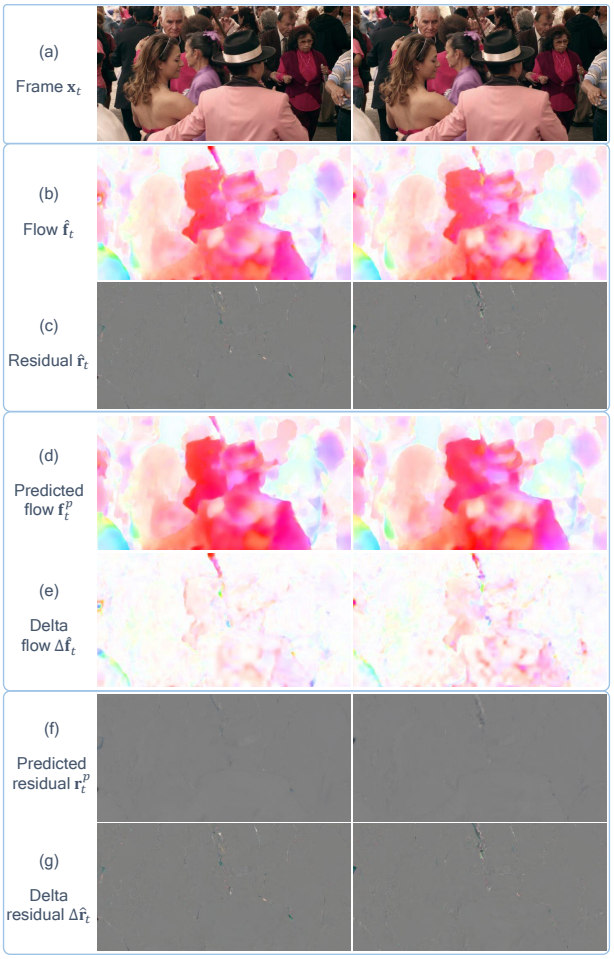

Figure 4: Illustration of video (a), flow (b) and residual (c), flow prediction and delta flow (d, e), and residual prediction and delta residual (f, g). Screenshot from Tango video from Netflix Tango in Netflix El Fuente* [36]

Figure 4 (d) and (f) show the predicted flow and residual, respectively. The associated delta flow and delta residual are shown in Figure 4 (e) and (g). As can be seen here, the predicted flow and residual maps are very close to the actual maps, leaving much less correlated delta flow and delta residual.

3.3. RGB and YUV420 architecture

Most literature on neural video codecs provides solutions for the RGB colorspace. Since the R, G, and B channels share the same resolution, working with RGB videos is trivial as the networks can conveniently concatenate them in the network input and output. However, standard codecs were designed to operate with YUV420 input mainly due to the resemblance with human vision. Since the chroma channels in the YUV420 colorspace are in a subsampled space, some

*Video produced by Netflix, with CC BY-NC-ND 4.0 license: https://media.xiph.org/video/derf/ElFuente/Netflix_Tango_Copyright.txt

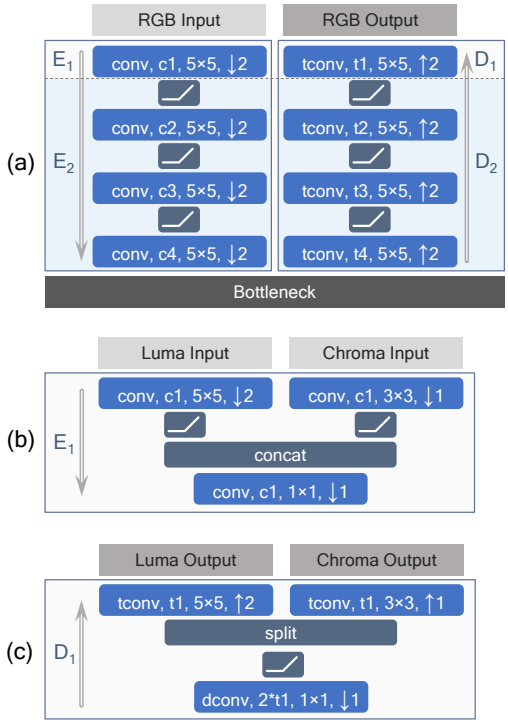

Figure 5: a) Network architecture for the RGB colorspace, b) YUV420 input head, c) YUV420 output head. `c*` and `t*` indicate the number of filters in the convolutional and transposed-convolutional layers, respectively.

modifications to the network architecture are needed to accommodate YUV420 videos. Figure 5 (a) shows the general architecture we used in the predictors and the autoencoders of our network for the RGB colorspace. The input is parsed by an encoder input head E1 and passed through an encoder backbone E2. The decoder consists of a backbone D2 and a decoder output head D1. At the bottleneck, the predictors simply pass the encoder outputs directly to the decoder. Components that transmit information will quantize the latents before passing these to the decoder.

To enable the codec with accommodating YUV420 inputs/outputs, we update the input/output heads E1/D1 in all the components of the network as shown in Figure 5 (b)/(c). E1 and D1 handle the luma (Y) and chroma (U and V) channels separately, and only the luma channels are down- or upsampled. Lastly, for the YUV420 codec, warping is applied for the luma and chroma channels separately. This ensures warping is always performed on the highest available spatial resolution.

4. Experiments

4.1. Network architecture

All of the components in our codec follow the architecture shown in Figure 5. The the number of filters in convolutional and transposed-convolutional layers is specified by `c1`, ..., `c4` and `t1`, ..., `t4`, respectively. More details are provided in Table 1. In the main autoencoders – including Flow-AE, Res-AE, and Image-AE (used for intra-frame coding) – the prior model follows the SSF design.

Module	`c1` & `t2`	`c2` & `t3`	`c3` & `t4`	`c4`
Image-AE	128	128	128	192
Flow-AE	128	128	128	192
Res-AE	128	128	128	192
Flow-Pred	16	32	64	128
Res-Pred	16	32	64	128

Table 1: Network architecture details. For all components, `t1` depends on the number of output channels.

	Module	Parameters		KMACs / pixel	
RGB	Image-AE	9.414M	30.4%	198.2	30.6%
	Flow-AE	10.038M	32.4%	210.2	32.4%
	Res-AE	10.028M	32.5%	207.8	32.0%
	Flow-Pred	0.748M	2.4%	16.2	2.5%
	Res-Pred	0.748M	2.4%	16.2	2.5%
YUV420	Image-AE	9.496M	30.4%	208.0	30.5%
	Flow-AE	10.124M	32.4%	225.7	33.0%
	Res-AE	10.111M	32.3%	217.6	31.9%
	Flow-Pred	0.751M	2.4%	16.3	2.4%
	Res-Pred	0.750M	2.4%	15.3	2.2%

Table 2: Parameters and MACs per pixel of our RGB (top) and YUV420 (bottom) codecs.

We report the computational complexity of our codec in Table 2, where we show number of parameters and the number of multiply-and-accumulate operations (MACs) per module. The MAC count is normalized by number of input pixels to enable comparison across input modalities and resolutions. We also show the corresponding percentage to put these numbers in context.

We emphasize that for each I-frame, only the Image-AE is used, and this model is not used for P-frames. For example, for a sequence of length 10, the total number of MACs for the RGB codec would be $1 \cdot 198.2 + 9 \cdot (210.2 + 207.8 + 16.2 + 16.1)$ KMACs per pixel.

Using MACs as a proxy for computational complexity, we observe that receiver-side predictors incur a small computational cost of only 2.5% of the entire model. Additionally, for each P-frame, both predictors incur $12\times$ fewer MACs than the P-frame autoencoders. Although the YUV420 input has fewer dimensions than the RGB input due to the subsampled chroma components, the fact that the YUV420 codec processes luma and chroma channels using separate layers results in a higher parameter count and MAC count than that of the RGB codec.

4.2. Dataset

Training is performed on the Vimeo90K dataset [37], a diverse set of 89,800 video sequences of resolution 256×448 in RGB. We evaluate on three common video compression benchmark datasets: UVG [20], MCL-JCV [33], and class B of HEVC (Common Test Conditions) [5], all available in raw YUV420 format. UVG contains 7 full-HD (*i.e.*, 1920×1080) videos at framerates of up to 120 fps. MCL-JCV contains 30 full-HD videos at 30 fps. HEVC class B contains 5 full-HD videos at various framerates.

4.3. Training

We train codecs with batch size of 8, where each sequences has 4 frames of size 256×256, resulting in one I-frame and three P-frames. We optimize our codec on MSE and MS-SSIM. Specifically, we initially train all models for 1,000,000 gradient updates on MSE, then trained the MS-SSIM models for extra 200,000 gradient updates on MS-SSIM, and finally finetuned all the models for 100,000 gradient updates. Training is done on 256×256 patches with learning rate 10^{-4}, and finetuning is done on 256×384 patches with learning rate 10^{-5}, using the Adam optimizer.

For the RGB codec training, Video90k is used in the native RGB colorspace, and both MSE and MS-SSIM are averaged over the R, G, and B channels.

For the YUV420 codec training, Video90k is converted to YUV420 using ffmpeg, and we weigh the MSE and MS-SSIM for the luma component by a factor $\frac{6}{8}$ and the chroma components by a factor $\frac{1}{8}$, consistent with the evaluation procedure used for standard codecs [26].

4.4. Evaluation

We evaluate all models in terms of rate-distortion performance. The PSNR metric in RGB weighs all color-channels equally. For YUV420 input, the luma channel is weighed by a factor $\frac{6}{8}$, and both chroma channels by a factor $\frac{1}{8}$. This is consistent with the way this metric is used in standard codecs [26]. As there is no commonly agreed-upon way to measure MS-SSIM for YUV420 inputs, we simply follow the same procedure as for PSNR, re-weighting the MS-SSIM for the luma and chroma components.

As the evaluation datasets are available in raw YUV420 format, they are used in the native colorspace for the YUV420 codec evaluation, see Figure 6 (a). However, evaluating in RGB requires a conversion from YUV420 to RGB using ffmpeg as shown in Figure 6 (b).

Most standard codecs were designed to work in the YUV420 input domain. To enable a fair comparison to standard codecs, we always feed them YUV420 inputs and always operate them in the YUV420 colorspace. The YUV420 output is converted to the target colorspace (if needed) as shown in Figure 6 (a) and (b) for comparison with our codecs in YUV420 and RGB, respectively.

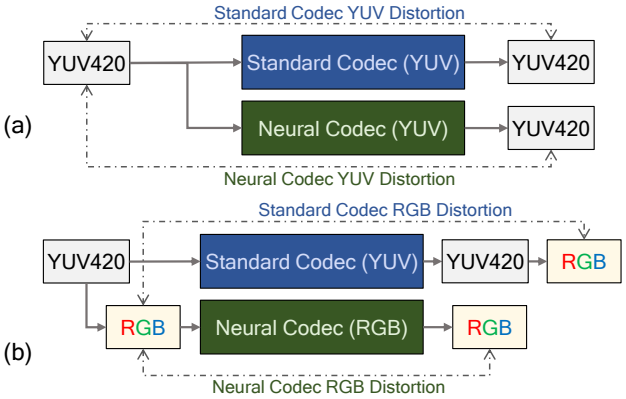

Figure 6: Evaluation procedure for YUV420 (a) and RGB (b). Color-space conversions are performed using ffmpeg.

4.5. Results

We measure rate-distortion performance on the UVG, MCL-JCV and HEVC-B datasets. We use PSNR and MS-SSIM as distortion measures, and measure bitrate in bits per pixel. We compare our RGB and YUV420 codecs to the reference implementation of HEVC called HM-16.25 [10] as well the the ffmpeg implementation of HEVC and AVC. We always use Group of Pictures size "infinite", even if some neural baselines use different GOP size. HM is used in the low-delay-P configuration, and ffmpeg is used with all the default parameters but with B-frames disabled. Please see appendix 6.2 for the used HM and ffmpeg commands. We compare our RGB codec with representative neural video coding methods including DVC [17], SSF [2], M-LVC [15], FVC [13], ELF-VC [23], RDAE [9], FRAE [8], and C2F [12]. The RGB and YUV420 rate-distortions results are shown in Figures 7 and 8, respectively. Here, SSF-YUV is a re-implementation of SSF with the required YUV420 modifications.

	UVG	HEVC-B	MCL-JCV
Average	-38.01%	-26.51%	-16.48%

Table 3: BD rate savings for RGB PSNR with with respect to the SSF base model. Lower is better.

	UVG	HEVC-B	MCL-JCV
Y	-28.34%	-28.50%	-21.85%
U	-47.32%	-44.38%	-34.63%
V	-55.68%	-43.72%	-36.51%
Average	-34.13%	-32.39%	-25.28%

Table 4: BD-rate savings for YUV420 PSNR with respect to the SSF-YUV model. For the average results, the luma and chroma BD-rates are weighed by $\frac{6}{8}$ and $\frac{1}{8}$ respectively. Lower is better.

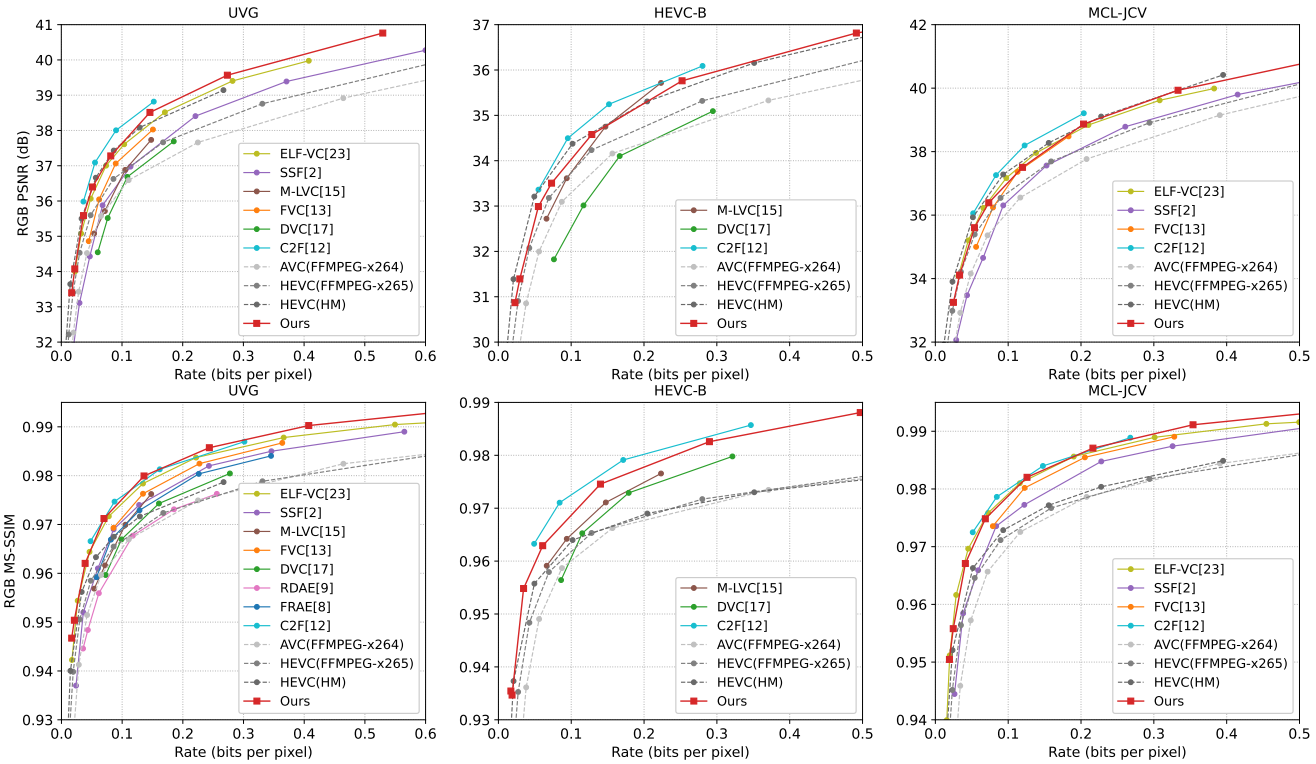

Figure 7: RGB rate-distortion curves for the UVG, HEVC Class-B, and MCL-JCV datasets.

In RGB colorspace, we outperform almost all of the compared methods in terms of both PSNR and MS-SSIM, except the very recent C2F [12]. In YUV420 colorspace, classical codecs dominate on YUV PSNR, but our codec obtains much better MS-SSIM on all the datasets.

Many modern neural codecs use motion compensation and residual coding, and we emphasize that our method is often applicable to those codecs as well. Here, we applied our method to the SSF architecture and therefore measured the performance improvement with respect to a (reimplemented) SSF and SSF-YUV baseline in Tables 3 and 4, respectively. We see that performance improvements are substantial, *e.g.*, more than 30% on the UVG dataset. Hence, although we are not generating state-of-the-art results in the RGB colorspace, our method has the potential to be added to a stronger base method and improve the state-of-the-art.

4.6. Ablations

To separate the effect of each component of our method, we perform ablation studies. We incrementally add the flow predictor, residual predictor, and connections between the bottleneck of the predictors and autoencoders (+conditions). The corresponding Bjontegaard-Delta [4] (BD)-rate gains are shown in Table 5. Each addition leads to strong BD-rate gains, and the flow predictor has the biggest effect for the high-framerate UVG dataset.

Component	UVG	HEVC-B	MCL-JCV
base model	0.0%	0.0%	0.0%
+flow predictor	-29.60%	-9.24%	-9.88%
+res predictor	-33.73%	-20.37%	-10.34%
+conditions	-38.01%	-26.51%	-16.48%

Table 5: BD rate savings for RGB PSNR with respect to the SSF base model. Lower is better.

Additionally, we measure bitrate savings for the flow and residual bitstreams individually to demonstrate that both Flow-AE and Res-AE benefit from our method. To measure BD-rate per component, we use the output PSNR as distortion and use the individual bitstream sizes per component as rate. The BD-rate gains versus PSNR for the UVG, HEVC class B, and MCL-JCV datasets are shown in Figure 9. Here, it can be observed that the predictors deliver significant rate savings on both flow and residual bitstreams across all PSNR values.

For completeness, we list two training techniques that did not work for us in early experiments and were therefore abandoned. Training with auxiliary losses for reconstructions resulting from flow and residual prediction, $\mathbf{x}_t^p$ and $\mathbf{x}_t^r$, did not work better than using only a distortion loss defined on the reconstructions $\hat{\mathbf{x}}_t$. Temporal loss modulation as in [23] resulted in $< 1\%$ rate savings, so we omitted it for simplicity.

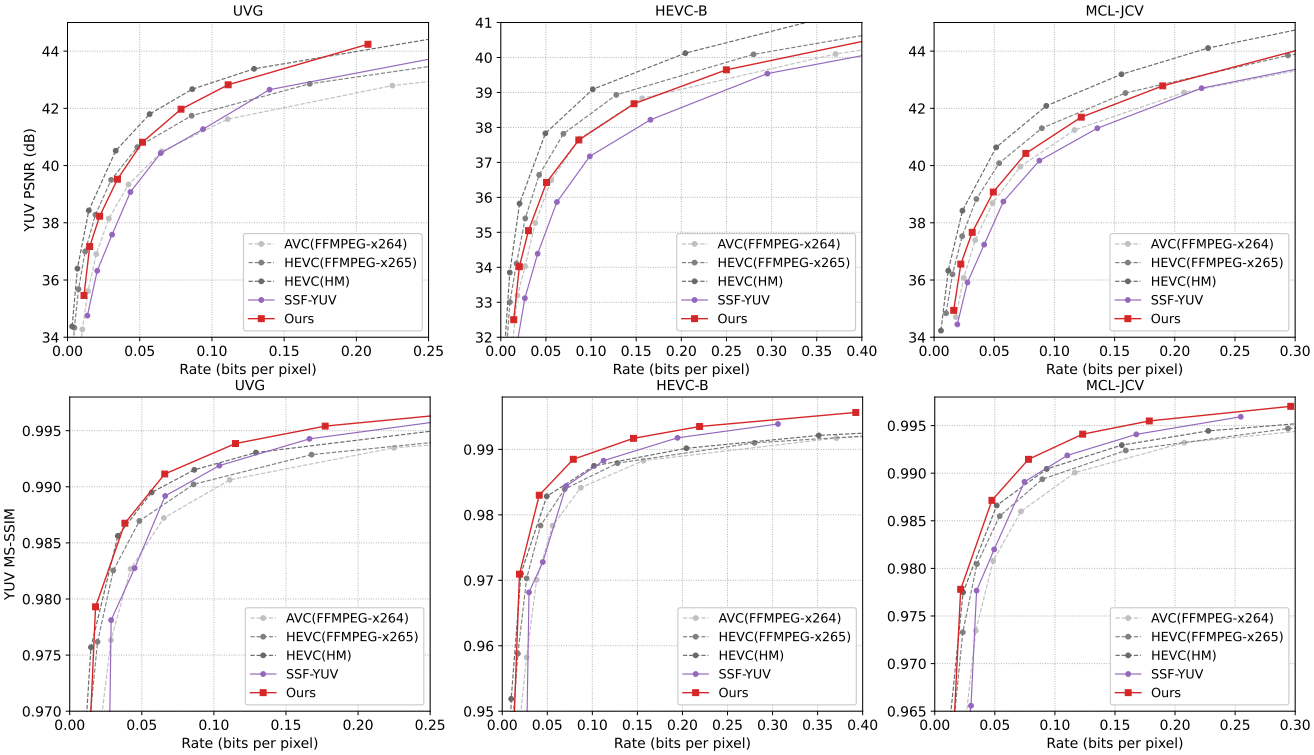

Figure 8: YUV420 rate-distortion curves for the UVG, HEVC Class-B, and MCL-JCV datasets.

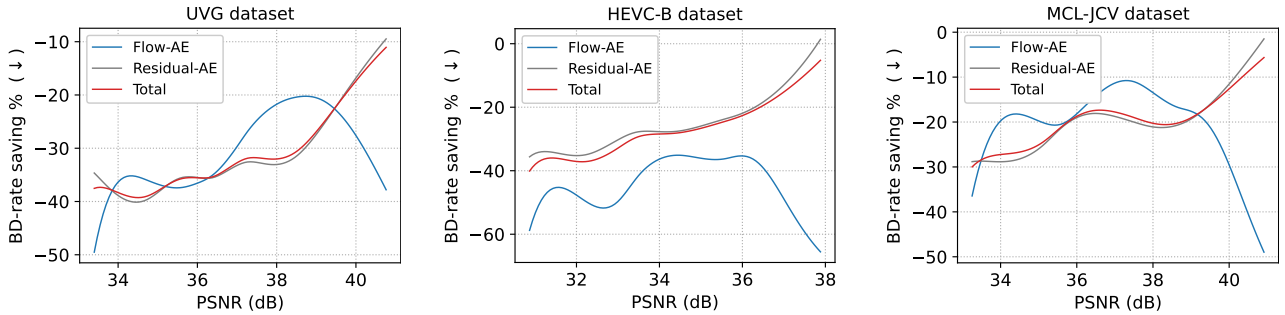

Figure 9: BD-rate savings versus RGB PSNR for flow and residual bitstreams and in-total for a SSF baseline. Lower is better.

5. Conclusion

In this work, we demonstrate that improved use of receiver-side information substantially improves the rate-distortion performance of neural video codecs. By having the decoder predict the flow and residual at a current timestep based on previously transmitted information, our codec only needs to transmit a correction, saving bits in the process. This results in Bjontegaard-Delta rate savings of up to 30% compared to a Scale-Space Flow baseline on common video benchmark datasets. The required architectural modification is straightforward, requires only lightweight components, and is applicable to other neural video codecs as well.

Furthermore, we train our model on both the RGB and YUV420 input domains with only minor architectural modifications, and show strong performance in both settings compared to neural and standard baselines. As standard codecs were designed to optimize YUV420 PSNR, our codec does not outperform HM on this metric in the low bitrate regime, but substantially outperform the HM codec on YUV420 MS-SSIM. Ablations show that both the flow and residual predictors contribute to the final performance.

Impact statement It is possible that learned codecs exacerbate biases present in the training data. Nevertheless, we believe that improved video coding efficiency has a net positive impact on the world by reducing bandwidth and storage space needs, and by improving visual quality for video applications such as video conferencing.

References

[1] *Low-Power Computer Vision: Improve the Efficiency of Artificial Intelligence (1st ed.)*. Chapman and Hall/CRC, 2022.

[2] 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.

[3] Eirikur Agustsson, Michael Tschannen, Fabian Mentzer, Radu Timofte, and Luc Van Gool. Generative adversarial networks for extreme learned image compression. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2019.

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

[5] Frank Bossen et al. Common test conditions and software reference configurations. *JCTVC-L1100*, 12(7), 2013.

[6] Cisco. Report: Where does the majority of internet traffic come from? 2017.

[7] Hilmi E. Egilmez, Ankitesh K. Singh, Muhammed Coban, Marta Karczewicz, Yinhao Zhu, Yang Yang, Amir Said, and Taco S. Cohen. Transform network architectures for deep learning based end-to-end image/video coding in subsampled color spaces. *IEEE Open Journal of Signal Processing*, 2:441–452, 2021.

[8] Adam Goliński, Reza Pourreza, Yang Yang, Guillaume Sautière, and Taco S. Cohen. Feedback recurrent autoencoder for video compression. *ACCV*, 2020.

[9] 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.

[10] HM. Hevc offical test model. `https://hevc.hhi.fraunhofer.de/`, 2021, 2021.

[11] 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.

[12] 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 conference on Computer Vision and Pattern Recognition*, 2022.

[13] 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.

[14] Théo Ladune, Pierrick Philippe, Wassim Hamidouche, Lu Zhang, and Olivier Déforges. Conditional coding for flexible learned video compression. *ICLR neural compression workshop*, 2021.

[15] Jianping Lin, Dong Liu, Houqiang Li, and Feng Wu. M-lvc: Multiple frames prediction for learned video compression. *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2020.

[16] Chao Liu, Heming Sun, Jiro Katto, Xiaoyang Zeng, and Yibo Fan. Learned video compression with residual prediction and loop filter. Aug. 2021.

[17] 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.

[18] Fabian Mentzer, Eirikur Agustsson, Johannes Ballé, David Minnen, Nick Johnston, and George Toderici. Neural video compression using gans for detail synthesis and propagation. *arXiv preprint arXiv:2107.12038*, 2021.

[19] Fabian Mentzer, George Toderici, Michael Tschannen, and Eirikur Agustsson. High-fidelity generative image compression. *Neural Information Processing Systems*, 2020.

[20] Alexandre Mercat, Marko Viitanen, and Jarno Vanne. Uvg dataset: 50/120fps 4k sequences for video codec analysis and development. In *Proceedings of the 11th ACM Multimedia Systems Conference*, MMSys '20, page 297–302, New York, NY, USA, 2020. Association for Computing Machinery.

[21] Markus Nagel, Marios Fournarakis, Rana Ali Amjad, Yelysei Bondarenko, Mart van Baalen, and Tijmen Blankevoort. A white paper on neural network quantization, 2021.

[22] Reza Pourreza and Taco S Cohen. Extending neural p-frame codecs for b-frame coding. *IEEE International Conference on Computer Vision*, 2021.

[23] 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.

[24] Ankitesh K Singh, Hilmi E Egilmez, Reza Pourreza, Muhammed Coban, Marta Karczewicz, and Taco S Cohen. A combined deep learning based End-to-End video coding architecture for YUV color space. Apr. 2021.

[25] Yannick Strümpler, Janis Postels, Ren Yang, Luc Van Gool, and Federico Tombari. Implicit neural representations for image compression. *arXiv preprint arXiv:2112.04267*, 2021.

[26] Jacob Ström, Kenneth Andersson, Rickard Sjöberg, Andrew Segall, Frank Bossen, Gary Sullivan, Jens-Rainer Ohm, and Alexis Tourapis. Working practices using objective metrics for evaluation of video coding efficiency experiments (draft 4). Input document JVET-T2016-v2, Joint Video Exploration Team (JVET) of ITU-T SG 16 WP 3 and ISO/IEC JTC 1/SC 29, Oct. 2020.

[27] 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.

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

[29] 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.

[30] George Toderici, Damien Vincent, Nick Johnston, Sung Jin Hwang, David Minnen, Joel Shor, and Michele Covell. Full resolution image compression with recurrent neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 5306–5314, 2017.

[31] 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.

[32] Ties van Rozendaal, Iris A. M. Huijben, and Taco S. Cohen. Overfitting for fun and profit: Instance-adaptive data compression, 2021.

[33] 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 *IEEE International Conference on Image Processing*, pages 1509–1513, 2016.

[34] Thomas Wiegand, Gary J Sullivan, Gisle Bjontegaard, and Ajay Luthra. Overview of the h. 264/avc video coding standard. *IEEE Transactions on circuits and systems for video technology*, 13(7):560–576, 2003.

[35] Chao-Yuan Wu, Nayan Singhal, and Philipp Krähenbühl. Video compression through image interpolation. *Proceedings of the European Conference on Computer Vision*, 2018.

[36] Xiph.org. Xiph.org video test media [derf's collection]. `https://media.xiph.org/video/derf/`. Accessed: 2020-02-21.

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

[38] Ren Yang, Luc Van Gool, and Radu Timofte. Perceptual learned video compression with recurrent conditional GAN. *arXiv preprint arXiv:2109.03082*, Sept. 2021.