Supplementary Material to Neural Video Compression with Diverse Contexts

Jiahao Li, Bin Li, Yan Lu
Microsoft Research Asia
{li.jiahao, libin, yanlu}@microsoft.com

This document provides the supplementary material to our proposed neural video compression (NVC) with diverse contexts, i.e., DCVC-DC model.

1. Network Structure

Our DCVC-DC is based on DCVC-HEM [9], but focuses more on exploiting Diverse Contexts to further boost compression efficiency. The learning of hierarchical quality pattern is mainly performed in training phase via adjusting the distortion weight in the loss. Here we describe other implementation details in network structure.

Group-based offset diversity. In this process, we will predict a total of $G \times N$ offsets d_t, where G is the group number and N is offset number for each group. In the implementation, the channel dimension of propagated feature F_{t-1} is 48. It is divided into 16 groups and each group has 2 offsets, i.e., G=16 and N=2. The detailed network structure of offset prediction is shown in Fig. 1. For convolution layer, the $(K, Cin, Cout, S)$ indicate the kernel size, input channel number, output channel number, and stride, respectively The inputs include the decoded motion vector (MV) $\hat{v}_t$. In addition, the previous reconstructed frame $\hat{x}_t$ and propagated F_{t-1} are also warped and fed as the auxiliary information. The outputs include the residual offsets d_t. d_t adds the $\hat{v}_t$ to get the final offsets o_t. In addition, the corresponding masks m_t are also generated. It is noted that, the first convolution layer will reduce the resolution by 2x for acceleration. After the last convolution layer, we use bilinear to upsample them back to original resolution.

Quadtree partition-based entropy coding. The proposed entropy coding can be classified into 4 steps. The network structure is shown in Fig. 2. As shown in this figure, the quantized latent representation $\hat{y}_{t-1}$ from the previous frame, hyper prior $\hat{z}_t$, and temporal context C_t are also used for predicting the distribution parameters for all $\hat{y}_t$ in the 4 steps. In addition, all positions coded in previous steps will also be used for predicting the distribution parameters of the positions coded in the current steps. In this process, to reduce the model parameters, most network blocks therein use the shared weights, as shown in Fig. 2.

Structure optimization. As mentioned in the main paper, to reduce the computation cost, we widely adopt the depthwise separable convolution. As shown in Fig. 2, the

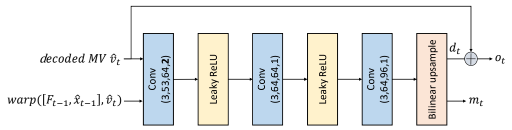

Figure 1. The network structure of offset prediction.

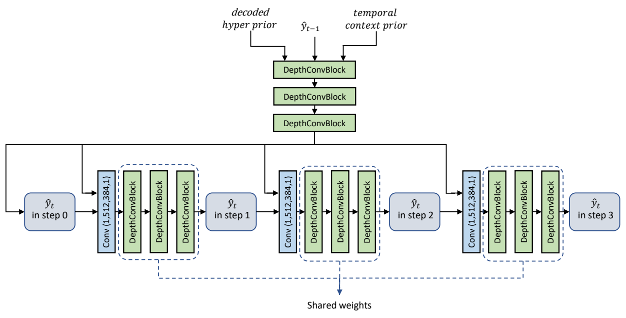

Figure 2. The network structure of entropy model.

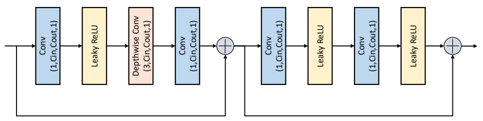

Figure 3. The network structure of DepthConvBlock.

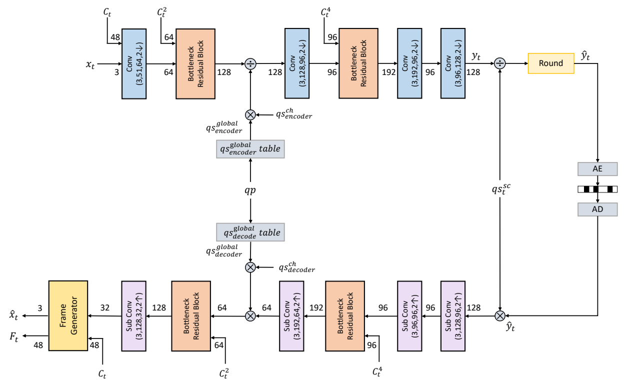

Figure 4. The network structure of contextual encoder and decoder. The frame generator follows the decoder.

basic block in our entropy model is DepthConvBlock which contains depthwise separable convolution. The structure of DepthConvBlock is shown Fig. 3. In the DepthConvBlock, except that the depthwise convolution layer is with 3x3 kernel, all regular convolution layers use 1x1 kernel to further reduce the computation cost.

In addition, we use the unequal channel number settings for the encoder and decoder. The network structure of our contextual encoder and decoder for frame coding is shown in Fig. 4. From this figure, we can see that the propagated feature F_t and the motion-aligned C_t with high resolution are with 48 channel number. They are smaller than 64 used in [9]. It can help us reduce the computation cost. At the same time, the quantized latent representation $\hat{y}_t$ uses 128 channel number, and it is larger than 96 used in [9]. This can bring some compression ratio improvements as the latent representation has larger capacity. By adjusting the channel number for features with different resolutions, a better trade-off between compression ratio and computation cost can be achieved. In Fig. 4, we follow [9, 12] and also adopt the multi-scale contexts C_t^2 and C_t^4, which are 2x and 4x down-sampled temporal contexts. Their generation details can be found in [9, 12]. The bottleneck residual block and frame generator in Fig. 4 are similar with those in [9].

Our codec supports variable bitrates in single model. For more precise rate adjustment, this paper proposes moving partial quantization operations to higher resolution. As shown in Fig. 4, the quantization parameter qp is used for controlling the bitrate via the user input. According to the qp, the global quantization step $qs_{encoder}^{global}$ is queried via the learnable quantization parameter-to-quantization step table. Then the learnable channel-wise $qs_{encoder}^{ch}$ is used to modulate the quantization step for each channel. For y_t at 16x down-sampled resolution, the spatial-channel-wise quantization step qs_t^{sc} is applied before the rounding operation, where the qs_t^{sc} for each frame is generated by the entropy model, like [9]. During the decoding, the corresponding inverse operations are applied. This multi-granularity quantization mechanism originates from [9]. However, in [9], global, channel-wise, and spatial-channel-wise quantization steps are all applied in the 16x down-sampled resolution. By contrast, we propose moving the global and channel-wise to the 2x down-sampled resolution for finer-grained adjustment. In addition, in our codec, the encoder and decoder have separate learnable global quantization step table and channel-wise quantization step, which further enlarges the flexibility. As shown in Fig. 5, we test 64 qs values for our codec. From the curve, we can see that our codec achieves very smooth rate adjustment in single model.

2. Test Settings

To conduct comprehensive comparisons, we compare the NVCs and traditional codecs in both YUV420 and RGB colorspaces. The test pipeline is shown in Fig. 6.

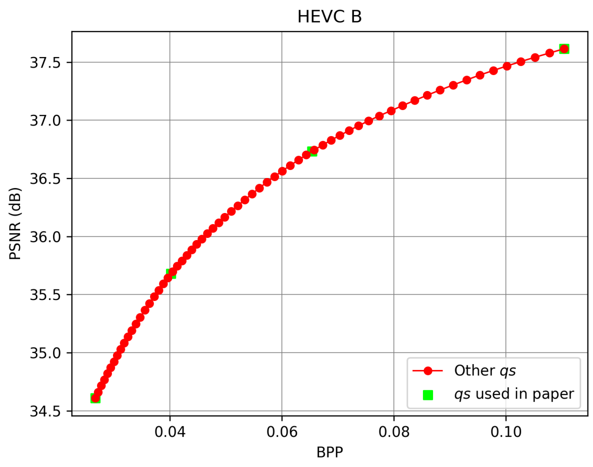

Figure 5. Smooth rate adjustment in single model.

YUV420. When testing YUV420 video, there is no any colorspace conversion, as shown in Fig. 6. For traditional codec, our benchmarks include HM [6], VTM [14], and ECM [4]. The three codecs use *encoder_lowdelay_main10.cfg*, *encoder_lowdelay_vtm.cfg*, and *encoder_lowdelay_ecm.cfg* config files, respectively. The parameters for each video are as:

- -c {*config file name*}
  --InputFile={*input video name*}
  --InputBitDepth=8
  --OutputBitDepth=8
  --OutputBitDepthC=8
  --FrameRate={*frame rate*}
  --DecodingRefreshType=2
  --FramesToBeEncoded={*frame number*}
  --SourceWidth={*width*}
  --SourceHeight={*height*}
  --IntraPeriod=32
  --QP={*qp*}
  --Level=6.2
  --BitstreamFile={*bitstream file name*}

RGB. Except that HEVC RGB testset is in RGB format, the raw formats of all other testsets are YUV420. Thus, to test RGB video, we need to convert them from YUV420 to RGB colorspace. Many existing NVC works use BT.601 (the default choice in FFmpeg) to conduct the conversion. Actually, JPEG AI [2, 3] uses BT.709 for the colorspace conversion. Thus, in the main paper, we follow JPEG and also use BT.709 to convert the raw YUV420 video to RGB colorspace when testing RGB video. In addition, it is

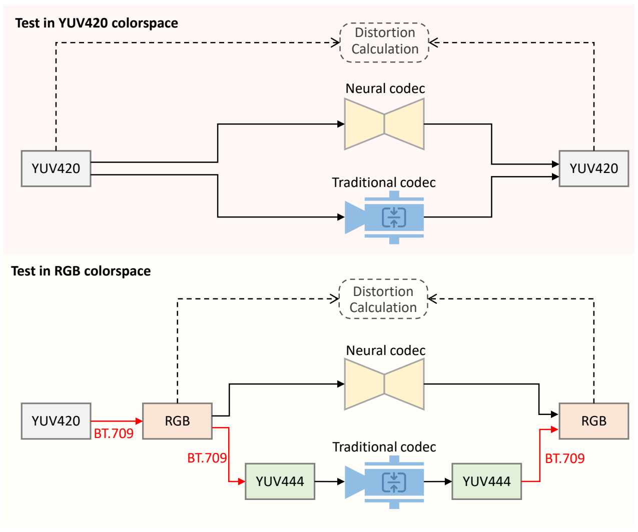

Figure 6. Test pipeline in YUV420 and RGB colorspace, respectively. The red line indicates the colorspace conversion.

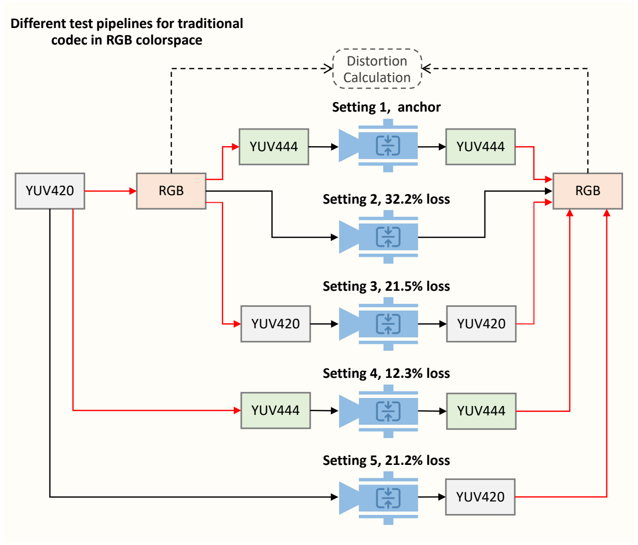

Figure 7. Comparison of different input colorspaces for traditional codec when testing RGB videos. For the bitrate comparison, VTM-17.0 is used and tested on HEVC B dataset.

also noted that the raw YUV420 videos of HEVC datasets [1, 5, 13] themselves are generated from RGB source using BT.709. However, it is unfortunate that currently we cannot access their raw RGB videos, and only have their YUV420 videos. Therefore, when testing RGB video, we should use the same conversion manner (i.e., BT.709) to convert them back from YUV420 to RGB.

It is noted that, when traditional codecs test RGB videos, using YUV444 as the internal colorspace achieves better compression ratio than directly using RGB, although the final distortion is measured in RGB. Fig. 6 shows that the RGB videos will be converted to YUV444 for higher compression ratio when testing traditional codecs. The reconstructed YUV444 videos will converted back to RGB for distortion calculation.

Fig. 7 compares different test pipelines for traditional codec in testing RGB videos. From this figure, we can see that using YUV444 as the internal colorspace (i.e., setting 1) achieves the best performance, and other settings have non-trivial bitrate increase. Many existing works use setting 5 in Fig. 7. However, we can see that there is 21.2% bitrate increase when compared with the setting 1 we used. Thus, to configure the best traditional codecs, we use YUV444 as the internal colorspace. For HM, VTM, and ECM, *encoder_lowdelay_main_rext.cfg*, *encoder_lowdelay_vtm.cfg*, and *encoder_lowdelay_ecm.cfg* config files are used, respectively. The parameters for each video are as:

- -c {*config file name*}
  --InputFile={*input file name*}
  --InputBitDepth=10
  --OutputBitDepth=10
  --OutputBitDepthC=10
  --InputChromaFormat=444
  --FrameRate={*frame rate*}
  --DecodingRefreshType=2
  --FramesToBeEncoded={*frame number*}
  --SourceWidth={*width*}
  --SourceHeight={*height*}
  --IntraPeriod=32
  --QP={*qp*}
  --Level=6.2
  --BitstreamFile={*bitstream file name*}

In addition, it is noted that ECM is still under development. As it is mainly optimized for YUV420, currently ECM-5.0 has several bugs on supporting YUV444 when using it to test RGB videos. We fixed them and verified the encoding and decoding match. After the bug fix, ECM-5.0 performs better than VTM-17.0, and the bitrate saving over VTM-17.0 is similar with that in YUV420. Thus, we believe the fix is reasonable for ECM-5.0 to support YUV444 coding.

3. Results in RGB colorspace with BT.601

Actually, when testing RGB videos, most existing NVC methods ignore the conversion manner and directly use BT.601 to conduct the conversion, because BT.601 is the default choice of FFmpeg. To make comparison with more existing NVCs, we also test our DCVC-DC under BT.601. It is noted that our DCVC-DC does not need any retraining

Table 1. BD-Rate (%) comparison for RGB colorspace with BT.601. Quality is measured with PSNR. The anchor is VTM-17.0.

	UVG	MCL-JCV	HEVC B	HEVC C	HEVC D	HEVC E	HEVC RGB	Average
VTM-17.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
VTM-13.2 (from [9])	8.8	6.3	3.2	1.9	0.5	8.6	5.5	5.0
HM-16.20 (from [9])	48.8	51.2	43.5	40.9	35.8	59.3	51.1	47.2
DVCPro [11]	238.1	176.1	194.8	204.5	157.9	455.1	179.2	229.4
MLVC [10]	137.6	140.0	126.0	216.7	165.7	262.2	163.5	173.1
RLVC [15]	244.0	221.3	205.1	202.7	141.8	398.2	199.4	230.4
CANF-VC [7]	61.4	60.5	56.4	70.5	52.8	119.7	79.9	71.6
DCVC [8]	140.3	107.2	117.9	151.5	106.7	269.5	111.9	143.6
DCVC-TCM [12]	29.9	39.4	32.7	62.4	27.8	80.4	24.4	42.4
DCVC-HEM [9]	–7.7	1.1	–1.1	16.9	–8.4	20.8	–9.9	1.7
Our DCVC-DC	–21.0	–13.3	–13.7	–8.2	–27.9	–14.4	–27.6	–18.0

Table 2. BD-Rate (%) comparison for RGB colorspace with BT.601. Quality is measured with MS-SSIM. The anchor is VTM-17.0.

	UVG	MCL-JCV	HEVC B	HEVC C	HEVC D	HEVC E	HEVC RGB	Average
VTM-17.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
VTM-13.2 (from [9])	3.4	4.6	2.6	1.8	0.5	13.1	3.9	4.3
HM-16.20 (from [9])	37.2	46.3	40.2	39.4	36.0	59.3	44.7	43.3
DVCPro [11]	72.3	43.5	64.5	61.6	24.3	248.1	67.8	83.2
RLVC [15]	86.2	77.6	68.5	79.5	35.0	311.8	68.0	103.8
CANF-VC [7]	31.2	14.2	30.7	26.3	11.4	160.8	57.7	47.5
DCVC [8]	37.1	10.2	33.8	25.5	2.2	158.4	38.4	43.7
DCVC-TCM [12]	–7.5	–19.3	–21.5	–21.1	–36.2	12.6	–22.2	–16.5
DCVC-HEM [9]	–32.6	–42.7	–45.7	–42.5	–54.5	–28.2	–43.6	–41.4
Our DCVC-DC	–37.5	–49.4	–53.4	–54.0	–63.1	–49.7	–54.4	–51.6

for testing RGB videos with BT.601. Table 1 and 2 show the BD-rate comparisons in terms of PSNR and MS-SSIM, respectively.

In this two tables, we use a newer version of VTM, i.e., VTM-17.0 as the anchor, when compared with the VTM-13.2 used in [9]. At the same time, we use the 10 bit intermediate representation for YUV444 rather than 8 bit used in [9]. These two modifications brings a more powerful baseline. As shown in Table 1, the VTM-13.2 used in [9] has an average 5.0% bitrate increase than VTM-17.0 used in this paper.

When using the stronger VTM-17.0 as anchor, we can see that our DCVC-DC also achieves significant bitrate saving for RGB videos converted using BT.601. For example, Table 1 shows that our DCVC-DC can achieve an average of 18.0% bitrate saving over VTM-17.0. By contrast, other NVCs still cannot surpass VTM-17.0. These results verify the effectiveness of our DCVC-DC.

4. Rate-Distortion Curves

In this document, we show the rate-distortion (RD) curves of all datasets, which correspond to the results in the main paper. Fig. 8 and 9 show the RD curves for videos in RGB colorspace with BT.709. Fig. 10 shows the RD curves for videos in YUV420 colorspace without any conversion. From these figures, we can see that our DCVC-DC can achieve SOTA compression ratio in a wide bitrate range.

5. Visual Comparison

Here we also provide some visual comparisons to demonstrate the advantage of our codec. Fig. 11 shows four examples. From these examples, we can see that our DCVC-DC can reconstruct clearer textures without increasing the bitrate cost, when compared with VTM-17.0 and ECM-5.0. In addition, it is also noted that, despite we learn the hierarchical quality pattern, there is no visual flicker in the decoded video. As shown in the PSNR curve in the main

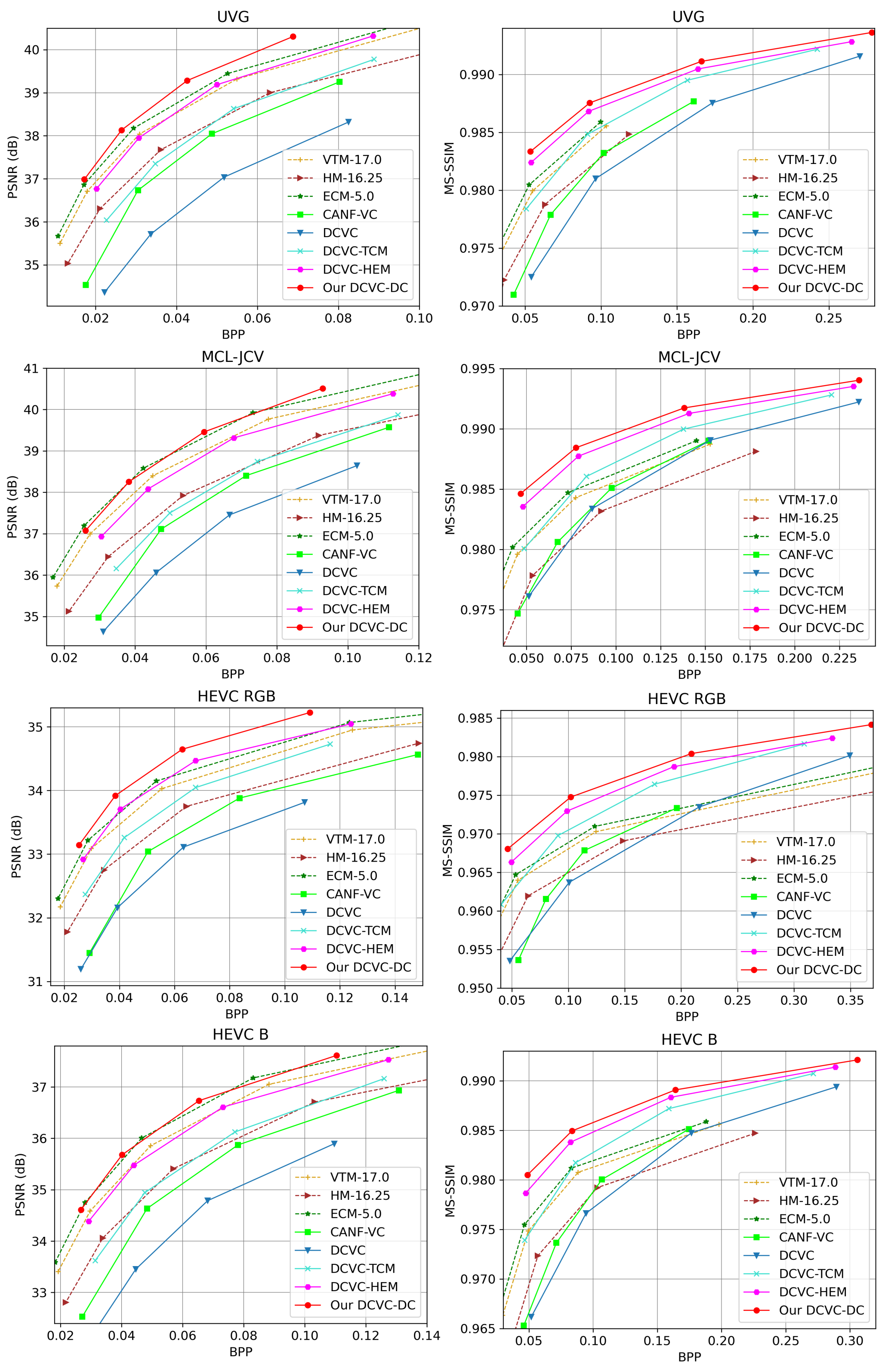

Figure 8. RD curves of UVG, MCL-JCV, HEVC RGB and B. The comparison is in RGB colorspace with BT.709. The left column is with PSNR and right column is with MS-SSIM.

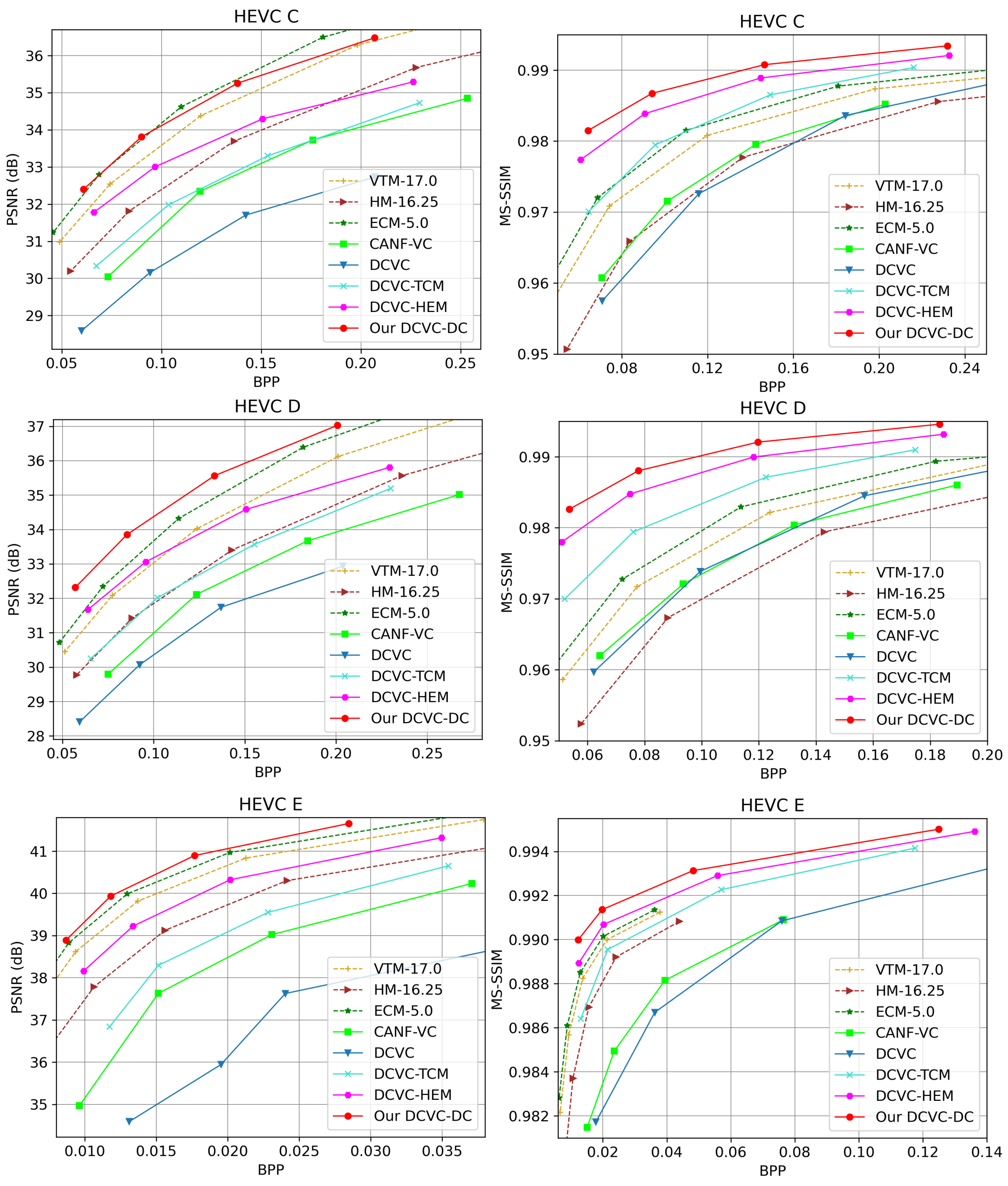

Figure 9. RD curves of HEVC C, D, and E. The comparison is in RGB colorspace with BT.709. The left column is with PSNR and right column is with MS-SSIM.

paper, we can see that our DCVC-DC actually has smaller PSNR variance than VTM-17.0. The standard community has verified the hierarchical quality pattern can improve the compression ratio with negligible visual degradation.

References

[1] 1080p, WVGA, WQVGA video coding test sequences. https://www.itu.int/wftp3/av-arch/video-site/0906_LG/VCEG-AL16.zip. Accessed: 2022-11-02. 3

[2] E. Alshina, J. Ascenso, T. Ebrahimi, F. Pereira, and T. Richter. [AHG 11] Brief information about JPEG AI CfP status. In *JVET-AA0047*, 2022. 2

[3] Anchors · JPEG-AI MMSP Challenge. https://jpegai.github.io/7-anchors/, 2022. Accessed: 2022-11-02. 2

[4] ECM-5.0. https://vcgit.hhi.fraunhofer.de/ecm/ECM, 2022. Accessed: 2022-11-02. 2

[5] Oelbaum et. al. The SVT High Definition Multi Format Test Set. In *ISO/IEC JTC1/SC29/WG11 M 13874, October 2006, Hangzhou, China.* 3

[6] HM-16.25. https://vcgit.hhi.fraunhofer.de/jvet/HM/, 2022. Accessed: 2022-11-02. 2

[7] Yung-Han Ho, Chih-Peng Chang, Peng-Yu Chen, Alessan-

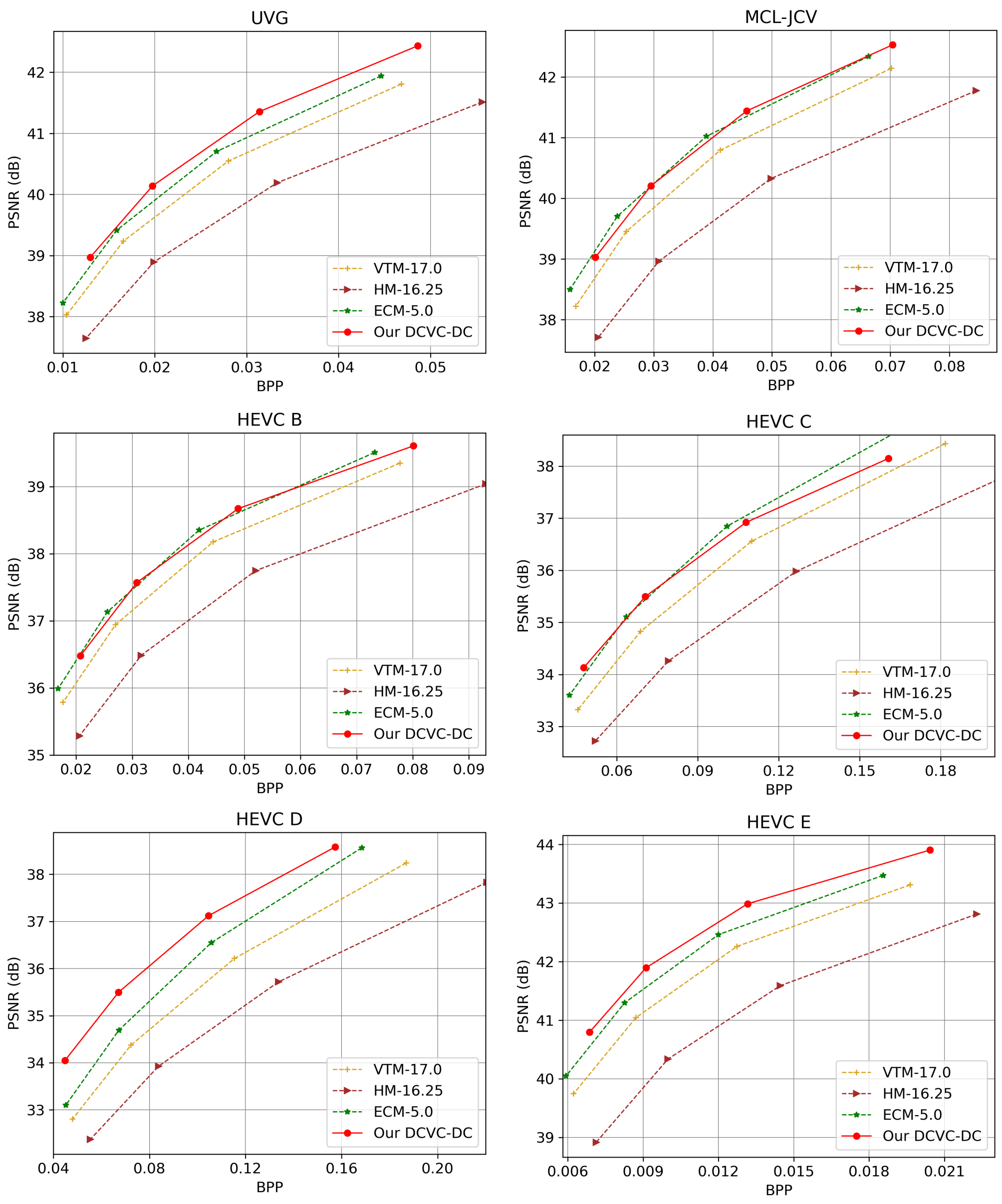

Figure 10. RD curves of UVG, MCL-JCV, HEVC B, C, D, and E. The comparison is in YUV420.

dro Gnutti, and Wen-Hsiao Peng. Canf-vc: Conditional augmented normalizing flows for video compression. *European Conference on Computer Vision*, 2022. 4

[8] Jiahao Li, Bin Li, and Yan Lu. Deep contextual video compression. *Advances in Neural Information Processing Systems*, 34, 2021. 4

[9] 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, 4

[10] Jianping Lin, Dong Liu, Houqiang Li, and Feng Wu. M-LVC: multiple frames prediction for learned video compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020. 4

[11] Guo Lu, Xiaoyun Zhang, Wanli Ouyang, Li Chen, Zhiyong Gao, and Dong Xu. An end-to-end learning framework for video compression. *IEEE transactions on pattern analysis and machine intelligence*, 2020. 4

[12] Xihua Sheng, Jiahao Li, Bin Li, Li Li, Dong Liu, and Yan Lu. Temporal Context Mining for Learned Video Compression. *IEEE Transactions on Multimedia*, 2022. 2, 4

[13] The SVT High Definition Multi Format Test Set . https://tech.ebu.ch/docs/hdtv/svt-multiformat-conditions-v10.pdf. Accessed: 2022-11-02. 3

[14] VTM-17.0. https://vcgit.hhi.fraunhofer.de/jvet/VVCSoftware_VTM/, 2022. Accessed: 2022-11-

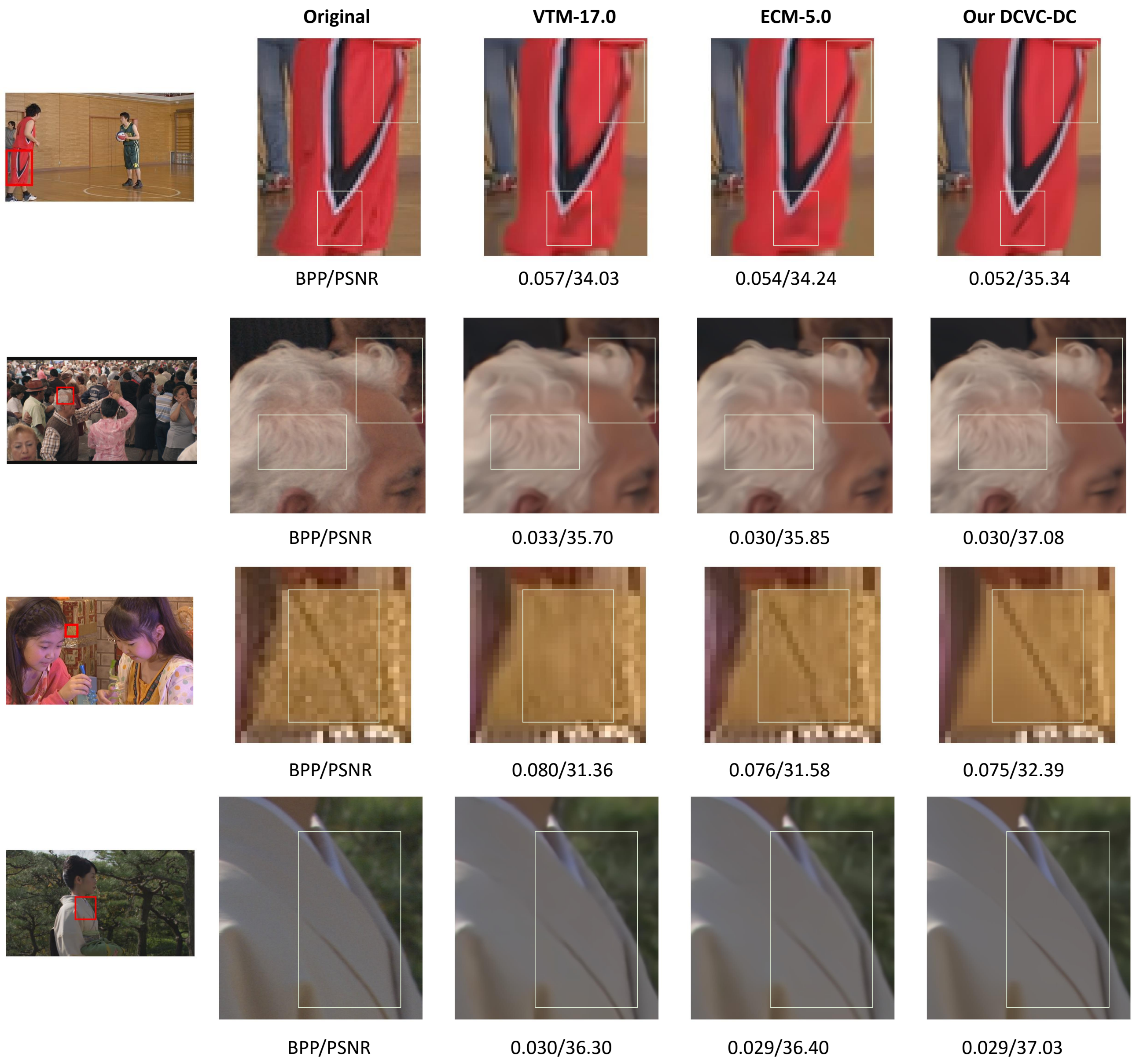

Figure 11. Visual comparison.

02. 2

[15] Ren Yang, Fabian Mentzer, Luc Van Gool, and Radu Timofte. Learning for video compression with recurrent auto-encoder and recurrent probability model. *IEEE Journal of Selected Topics in Signal Processing*, 15(2):388–401, 2021. 4