Supplementary Material to Towards Practical Real-Time Neural Video Compression

Zhaoyang Jia[1*] Bin Li[2] Jiahao Li[2] Wenxuan Xie[2] Linfeng Qi[1*] Houqiang Li[1] Yan Lu[2]
[1] University of Science and Technology of China [2] Microsoft Research Asia
{jzy_ustc, qlf324}@mail.ustc.edu.cn, lihq@ustc.edu.cn
{libin, li.jiahao, wenxie, yanlu}@microsoft.com

1. Test Settings

For a fairer comparison with traditional codecs, we employ their best settings to represent their best compression ratio. We test them in both YUV420 and RGB colorespaces for a comprehensive comparison.

YUV420 colorspace. We focus our comparison on the YUV420 colorspace, which is commonly optimized in traditional video codecs. Our evaluation includes the comparison with HM [2], VTM [3], and ECM [1], representing the best H.265 encoder, the best H.266 encoder, and the prototype of the next-generation traditional codec, respectively. For each traditional codec, we utilize the officially provided config files: *encoder_lowdelay_main10.cfg*, *encoder_lowdelay_vtm.cfg*, and *encoder_lowdelay_ecm.cfg*. The parameters for encoding 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={*intra period*}
  --QP={*qp*}
  --Level=6.2
  --BitstreamFile={*bitstream file name*}

RGB colorspace. In our experiments, the raw videos are stored at YUV420 format. So we convert them from YUV420 to RGB colorspace for testing. Following JPEG AI [4, 5] and [10, 11], we utilize BT.709 to perform this conversion, which brings higher compression ratio compared to the commonly-used BT.601. We test traditional codecs using 10-bit YUV444 as the internal colorspace and evaluate the final results in RGB. [10, 11] prove that, for traditional codecs, this brings better compression ratio than the direct measurement in RGB. For HM, VTM, and ECM, we utilize *encoder_lowdelay_rext.cfg*, *encoder_lowdelay_vtm.cfg*, and *encoder_lowdelay_ecm.cfg* as the config file, respectively. The parameters for coding 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={*intra period*}
  --QP={*qp*}
  --Level=6.2
  --BitstreamFile={*bitstream file name*}

2. Implementation Details

2.1. Module Structures

DCVC-RT follows a conditional coding manner [7, 8, 10, 11]. While the main paper covers the core network structure,

*This work was done when Zhaoyang Jia and Linfeng Qi were full-time interns at Microsoft Research Asia.

†This paper is the outcome of an open-source project started from Dec. 2023.

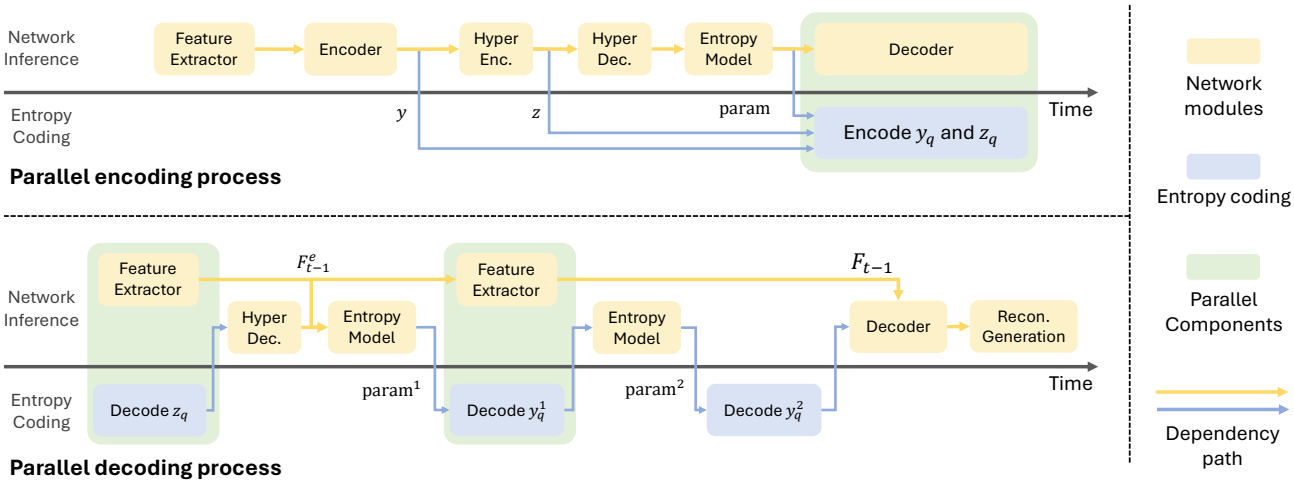

Figure 1. Encoding and decoding pipeline. y_q and z_q represent the symbols to be encoded in entropy coding. This parallel coding approach results in an average **12%** speedup in our encoding process and a **9%** speedup in our decoding process.

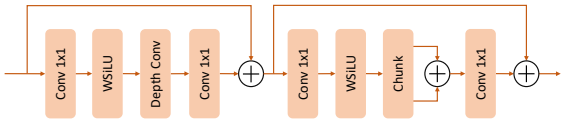

Figure 2. Structures of depth-wise convolution block (DC Block). The formulation of WSiLU is detailed in Section. 2.1. Chunk denotes splitting latents into two parts along the channel dimension.

we additionally illustrate the architecture of the depth-wise convolution block (DC Block) in Fig. 2, which is not detailed in the manuscript. Here, WSiLU denotes a weighted SiLU function [6], formulated as:

$$\text{WSiLU}(x) = x \cdot \text{Sigmoid}(\alpha \cdot x) \tag{1}$$

where the weighting parameter α is set to 4 by default.

2.2. Entropy model

DCVC-RT adopts a two-step distribution estimation scheme [9] to balance the coding speed and compression ratio. Although it results in a slight performance drop compared to more complex entropy models [10, 12], we adopt it since it enables significantly faster coding speed.

2.3. Parallel Coding

In practical scenarios, coding speed is influenced by both model inference time and entropy coding time. For DCVC-RT, encoding a 1080p frame typically requires around 7.8 ms for network inference and up to 2.6 ms for entropy coding, indicating that entropy coding significantly contributes to overall latency.

To accelerate the process, we propose a parallel coding scheme, depicted in Fig. 1. Observing that certain network

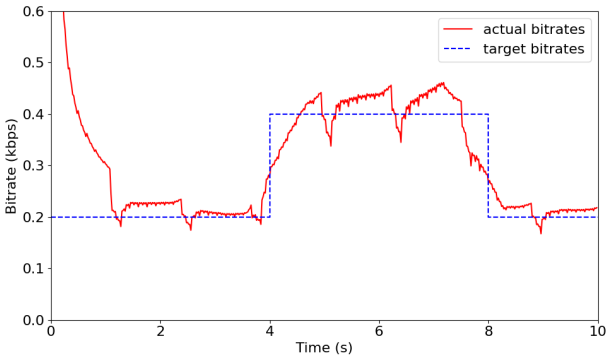

Figure 3. An example for rate-control on *Cactus* sequence.

modules can be inferred independently of entropy coding, we conduct these modules with entropy coding in parallel. This parallelization does not cause severe hardware resource contention, so the latency can be effectively reduced. For instance, we can perform entropy coding on the CPU without affecting concurrent network inference process on the GPU.

In encoding, the feature extractor, encoder, and entropy model are sequentially inferred, generating symbols y_q and z_q for entropy coding and the quantized latents for decoder inference. Since the coding of y_q and z_q does not depend on decoder inference, these processes are parallelized. It is worth noting that, as the reference feature is buffered before reaching the reconstruction module, the reconstruction generation module does not typically need to be inferred. In this case, our encoding is usually faster than decoding.

In decoding, the entropy decoding of z_q is independent of the feature extractor, allowing us to process them concurrently. Since decoding z_q is fast, we only infer the first

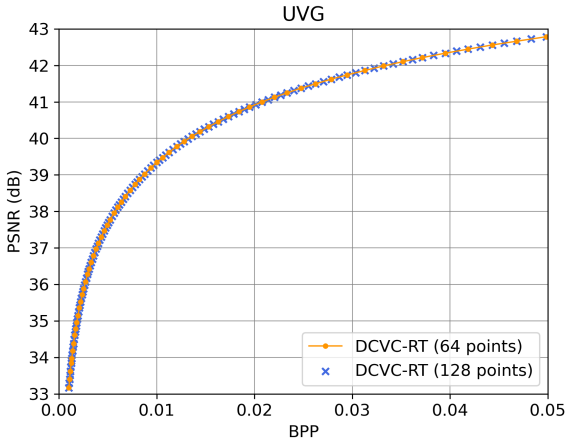

Figure 4. Extend supported rate number from original 64 rates to 128 rates using interpolation on UVG dataset.

part of the feature extractor at this stage. After probability estimation, the remainder of the feature extractor is inferred alongside the entropy decoding of y_q. Notably, we adopt a two-step coding scheme, where we first decode y_q^1 followed by y_q^2. Experiments show that the decoding time of y_q^1 covers the time needed for feature extractor inference.

In our implementation, the parallel coding approach results in an average 12% speedup in the encoding process and a 9% speedup in the decoding process, demonstrating its effectiveness in accelerating DCVC-RT.

2.4. Model Integerization

We provide a detailed explanation to better understand the model integerization pipeline. As mentioned in Equation **??**, all int16 features are linearly mapped from its floating-point counterparts in the original model. Moreover, we also create linear mappings for convolution weights w and biases b. For a convolution layer, we have

$$w_i = \text{round}(K_2 \cdot w_f)$$
$$b_i = \text{round}(K_2 \cdot b_f)$$

where $K_2 = 8192$ in our implementation. Let x be the input feature and y be the output feature, we derive the integer convolution from the floating-point convolution as follows

$$y_f = \text{conv}(x_f, w_f) + b_f$$
$$\frac{y_i}{K_1} = \text{conv}(\frac{x_i}{K_1}, \frac{w_i}{K_2}) + \frac{b_i}{K_2}$$
$$y_i = \frac{\text{conv}(x_i, w_i) + b_i \cdot K_1}{K_2}$$

where $K_1 = 512$ as mentioned in Equation **??**.

The processes above are rewritten as an algorithmic workflow in Algorithm 1.

Algorithm 1 Model Integerization for a Convolution

Input: Input feature x; Convolution weight w_f and bias b_f; Hyperparameters K_1 and K_2.
Output: Output feature y_i.
if x is of type fp16 **then** ▷ The input frame
    $x_i \leftarrow round(K_1 \cdot x)$.to_int16()
else ▷ The remaining layers
    assert x is of type int16
    $x_i \leftarrow x$
end if
Get int16 weight: $w_i = \text{round}(K_2 \cdot w_f)$.to_int16()
Get int16 bias: $b_i = \text{round}(K_2 \cdot b_f)$.to_int16()
$a \leftarrow \text{conv}(x_i, w_i) + b_i \cdot K_1$
Return: $y_i \leftarrow \text{clip}(\frac{a}{K_2}, -32768, 32767)$

Table 1. Sequences used for ablation on implicit temporal modelling. MCL-JCV contains 30 sequences, we denote each sequence using their ID, e.g., 01 denotes sequence videoSRC01_1920x1080_30.

Motion Type	Sequences
Large Motion	02, 04, 05, 07, 08, 10, 11, 14, 17, 19, 20, 21, 22, 24, 26
Small Motion	01, 03, 06, 09, 12, 13, 15, 16, 18, 23, 25, 27, 28, 29, 30
Scene Change	04, 14, 19, 20, 21, 25, 26, 27, 28, 29

Table 2. Breakdown on per-module complexity for coding a 1080p frame on A100 GPU.

Metric	Encoder	Decoder	Feature Extractor	Entropy Model	Reconstruction Generation
Latency	1.4 ms	1.4 ms	2.4 ms	2.4 ms	2.2 ms
kMACs/pixel	30.1	34.0	53.0	31.5	56.4

3. Experimental Results

3.1. Ablation on Implicit Temporal Modelling

In Tab. 2. of the main paper, we conduct an ablation study to compare explicit motion estimation with implicit temporal modeling under different motion conditions. Motion amplitude is determined using a pretrained SEA-RAFT [13] model to calculate the average motion between consecutive frames, enabling categorization into large or small motion content. Scene changes are identified manually. Tab. 1 lists the sequences categorized by motion type.

3.2. Results on Rate-Control

Rate control is an essential feature for video codecs, particularly for applications like streaming and real-time communication. By adjusting the quantization parameter (qp), DCVC-RT achieves effective rate-control. Fig. 3 illustrates this capability, showcasing how DCVC-RT can modulate

Table 3. BD-Rate (%) comparison in YUV420 colorspace. All frames with intra-period=–1.

	UVG	MCL-JCV	HEVC B	HEVC C	HEVC D	HEVC E	Average
VTM-17.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
DCVC-FM (fp16)	−16.8	−8.0	−15.4	−30.2	−37.5	−20.2	−21.3
DCVC-RT (fp16)	−24.0	−14.8	−16.6	−21.0	−27.3	−22.4	−21.0
DCVC-RT (int16)	−21.0	−12.3	−14.8	−20.0	−26.4	−15.0	−18.3
DCVC-RT Large (fp16)	−31.1	−22.1	−27.7	−32.1	−37.7	−34.0	−30.8
DCVC-RT Large (int16)	−27.7	−19.6	−25.9	−31.1	−37.0	−24.2	−27.6

Table 4. BD-Rate (%) comparison in RGB colorspace. All frames with intra-period=–1.

	UVG	MCL-JCV	HEVC B	HEVC C	HEVC D	HEVC E	Average
VTM-17.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
HM-16.25	43.2	49.5	49.9	45.2	39.9	47.7	45.9
DCVC-DC	9.2	0.0	14.9	5.3	−7.8	87.7	18.2
DCVC-FM	−11.0	−1.3	−11.5	−26.6	−33.8	−15.4	−16.6
DCVC-FM (fp16)	−10.4	−1.1	−11.2	−26.5	−33.7	−12.1	−15.8
DCVC-RT (fp16)	−17.2	−6.8	−11.3	−15.8	−21.3	−11.4	−14.0
DCVC-RT (int16)	−13.1	−3.9	−9.2	−14.8	−20.4	−3.2	−10.8
DCVC-RT Large (fp16)	−25.0	−14.8	−22.9	−27.5	−32.4	−24.4	−24.5
DCVC-RT Large (int16)	−20.5	−11.9	−20.9	−26.5	−31.7	−13.5	−20.8

bitrates effectively.

3.3. Per-Module Complexity Analysis

Tab. 2 presents a per-module complexity analysis, detailing the average inference time and the corresponding MACs for each module. Note that since both our encoding and decoding pipeline both do not execute all modules (e.g., reconstruction generation is skipped during encoding), the sum of latencies does not equal the overall latency.

3.4. Interpolation for Arbitrary Rates

In the proposed rate-adjustment module bank, we learn 64 rate points to accommodate different rates within a single model. These modules include vectors for latent modulation and the factorized modules for coding z. In practice, we can expand the supported rate number to arbitrary larger by performing interpolation in the module bank. Specifically, to achieve a rate between the i-th and the $i+1$-th module, we perform linear interpolation between the two vectors to obtain an intermediate vector for latent modulation. For the factorized module, we directly select the nearest module for probability estimation. In Fig. 4, we evaluate it to achieve 128 different rates (64 original and 64 interpolated) in a single model, and it demonstrates a very smooth quality adjustment.

3.5. Model Integerization Results

Maintaining calculation consistency across different devices is critical for video codecs, as encoding-decoding inconsistencies can lead to errors in entropy coding and ultimately yield corrupted reconstructions. Fig. 5 (left) shows how nondeterministic floating-point calculations can accumulate errors over time, producing visible artifacts after around 30 frames. In DCVC-RT, model integerization addresses this issue by facilitating model integerization and enforcing deterministic integer calculations. As shown in Fig. 5 (right), this approach ensures cross-device consistency.

Tab. 3 and 4 present the rate-distortion performance of DCVC-RT in int16 mode. Compared to fp16, the int16 model exhibits only a minor BD-Rate decrease of approximately 3%, demonstrating its practicality and effectiveness for real-world applications.

3.6. Results on RGB Colorspace

In Tab. 4, we present the BD-rate comparison for the RGB format under all frame intra period –1 settings. As depicted in the table, DCVC-RT achieves an average 14.0% bits saving compared to VTM, which is comparable to 15.8% of DCVC-FM. It demonstrate the high rate-distortion performance of DCVC-RT.

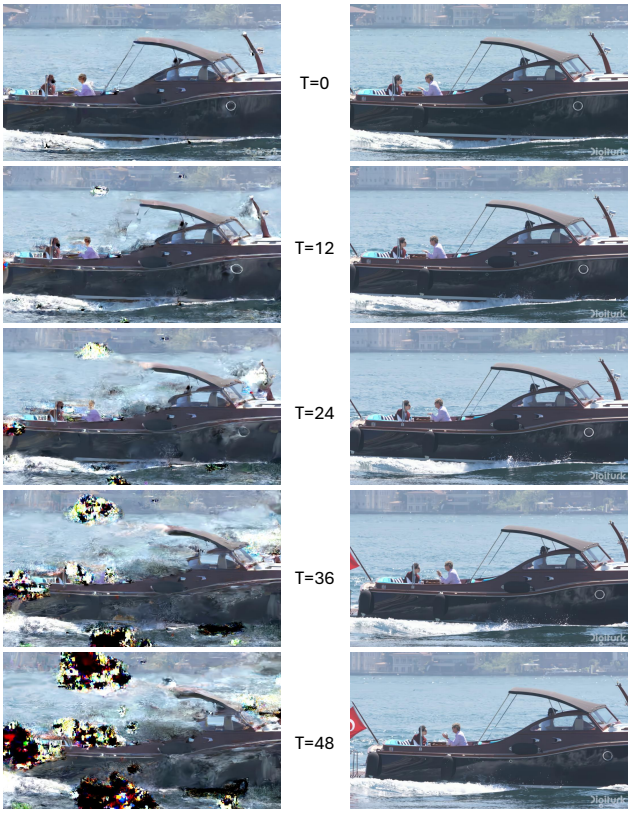

Figure 5. An example for cross-platform coding on *YachtRide* sequence. We perform encoding on an NVIDIA A100 GPU and perform decoding on an RTX 2080Ti.

3.7. Visual Comparison

To further demonstrate the superiority of DCVC-RT, we also present the visual comparison with VTM [3] and DCVC-FM [11] in Fig. 6. From these examples, we can observe that DCVC-RT can reconstruct frames with more details and clearer structures, even at lower bitrates. We offer video visualizations and comparisons on our project page: `https://dcvccodec.github.io/`.

3.8. Rate-Distortion Curves

In Fig. 7, we present the rate-distortion curves across all tested datasets. On the low-quality range, DCVC-RT demonstrates comparable or superior performance to DCVC-FM and surpasses ECM, showcasing its exceptional performance. However, we observe a performance decline at high rates. This decrease can be attributed to the lightweight design of DCVC-RT model, which has limited capabilities compared to larger models. To examine this problem, we increase the model capacity of DCVC-RT in Section. 3.9.

Table 5. Complexity analysis of DCVC-RT-Large. All methods are tested in fp16 mode on an A100 GPU. The coding speed are evaluated on 1080p videos. BD-Rate is calculated in YUV420 colorspace using VTM as anchor.

Model	Average BD-Rate	MACs	Coding Speed Enc.	Coding Speed Dec.
DCVC-FM	–21.3%	2642G	5.0 fps	5.9 fps
DCVC-RT-Large	–30.8%	812G	89.3 fps	73.4 fps
DCVC-RT	–21.0%	385G	125.2 fps	112.8 fps

3.9. Performance for a Large Model

In this paper, we primarily design a lightweight model to accelerate coding. However, DCVC-RT can be easily extended to a larger model (DCVC-RT Large) by increasing the number of channels and DCB blocks. As shown in Tables 3 and 4, the enhanced model capacity significantly improves compression performance. On YUV420 colorspace, Our large model, DCVC-RT-Large, achieves an average BD-Rate of -30.8%, outperforming -21.3% of the advanced large NVC model, DCVC-FM. On RGB colorspace, It achieves a BD-Rate of -24.5%, compared to DCVC-FM's -15.8%. Furthermore, the performance drop at high bitrates observed in the small model is effectively addressed in the large model, confirming the conclusion in Section. 3.8. For high-range rate points (calculated from $qp = 42$ to $qp = 63$), using DCVC-FM as the anchor, DCVC-RT shows a BD-Rate loss of 9.3% on UVG, whereas DCVC-RT-Large achieves a slightly better BD-Rate of -1.3%. Despite its improved performance, DCVC-RT-Large maintains significantly lower complexity than DCVC-FM, as shown in Table. 5. Thanks to its efficiency-driven design, it achieves an encoding speed of approximately 90 fps. These results underscore the superiority of DCVC-RT in the rate-distortion-complexity trade-off.

References

[1] ECM. `https://vcgit.hhi.fraunhofer.de/ecm/ECM`.

[2] HM. `https://vcgit.hhi.fraunhofer.de/jvet/HM`.

[3] VTM. `https://vcgit.hhi.fraunhofer.de/jvet/VVCSoftware_VTM`.

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

[5] Anchors · JPEG-AI MMSP Challenge. Anchors · JPEG-AI MMSP Challenge. `https://jpegai.github.io/7-anchors/`.

[6] Stefan Elfwing, Eiji Uchibe, and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural networks*, 107: 3–11, 2018.

[7] Yung-Han Ho, Chih-Peng Chang, Peng-Yu Chen, Alessandro Gnutti, and Wen-Hsiao Peng. CANF-VC: Conditional

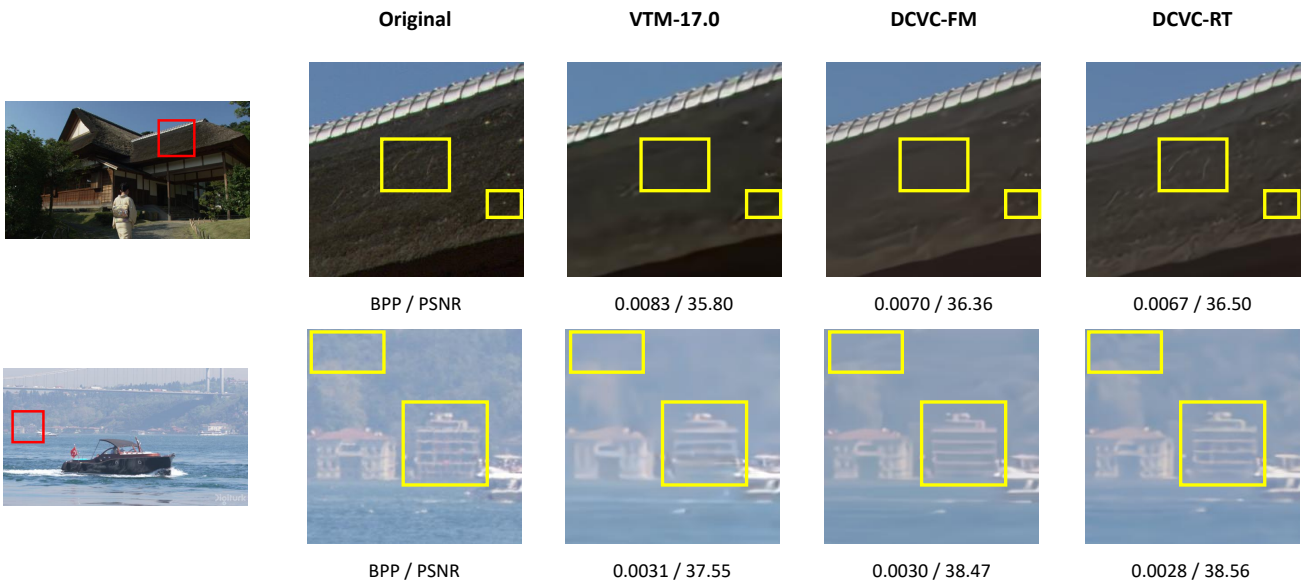

Figure 6. Visual comparison with VTM [3] and DCVC-FM [11]. *Best viewed when zoomed in.*

augmented normalizing flows for video compression. In *European Conference on Computer Vision*, pages 207–223. Springer, 2022.

[8] Jiahao Li, Bin Li, and Yan Lu. Deep contextual video compression. *Advances in Neural Information Processing Systems*, 34:18114–18125, 2021.

[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.

[10] Jiahao Li, Bin Li, and Yan Lu. Neural video compression with diverse contexts. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 22616–22626, 2023.

[11] Jiahao Li, Bin Li, and Yan Lu. Neural Video Compression with Feature Modulation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2024, Seattle, WA, USA, June 17-21, 2024*, 2024.

[12] 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.

[13] Yihan Wang, Lahav Lipson, and Jia Deng. SEA-RAFT: Simple, efficient, accurate raft for optical flow. In *European Conference on Computer Vision*, pages 36–54. Springer, 2025.

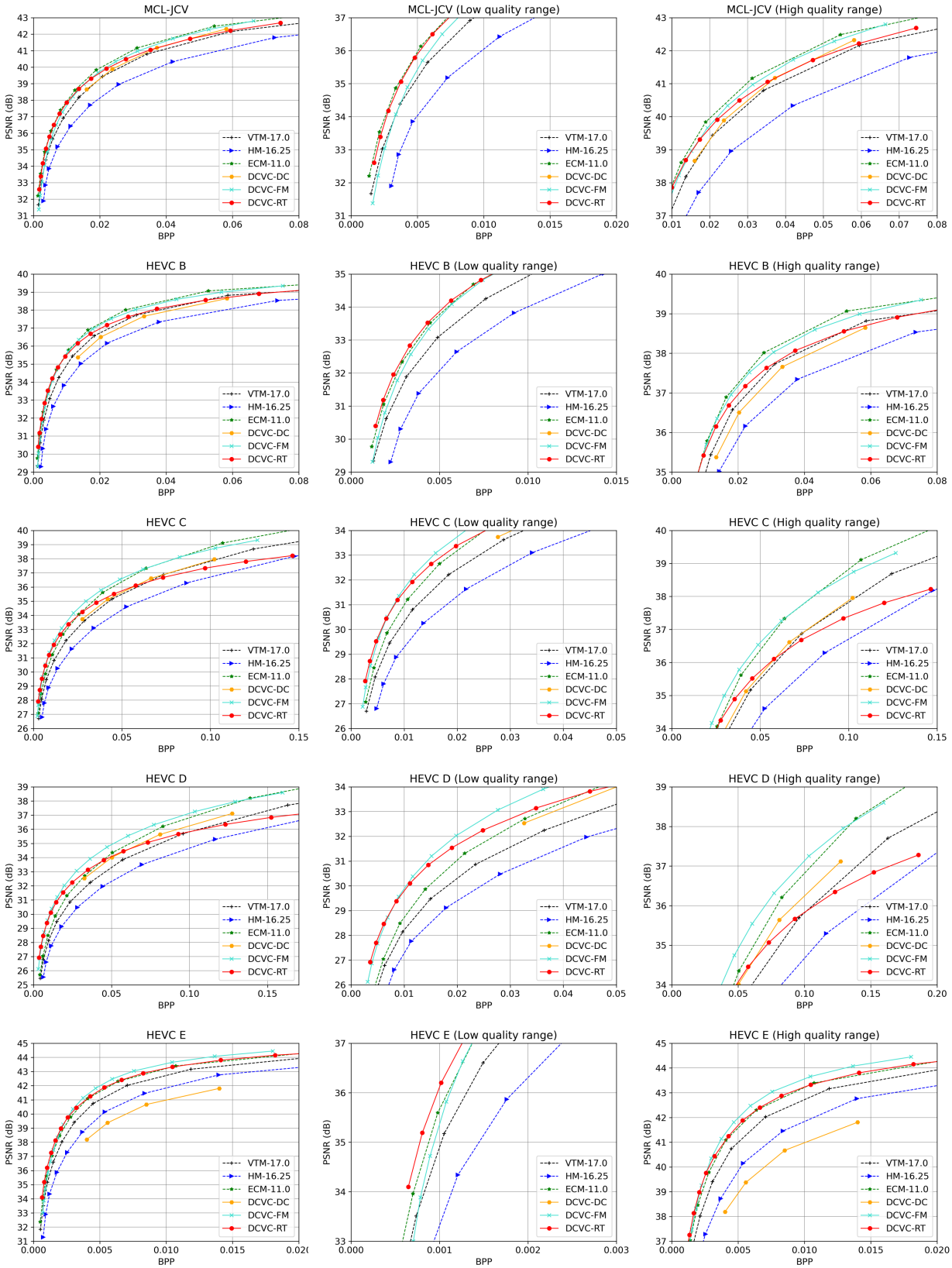

Figure 7. Rate-distortion curves for MCL-JCV and HEVC datasets. All frames are tested with intra-period=–1 in YUV420 colorspace. We show the whole quality range, relatively low quality range and relatively high quality range for each dataset.