Contextformer: A Transformer with Spatio-Channel Attention for Context Modeling in Learned Image Compression

Supplementary Material

A. Burakhan Koyuncu[1,2], Han Gao[3], Atanas Boev[2], Georgii Gaikov[2], Elena Alshina[2], and Eckehard Steinbach[1]

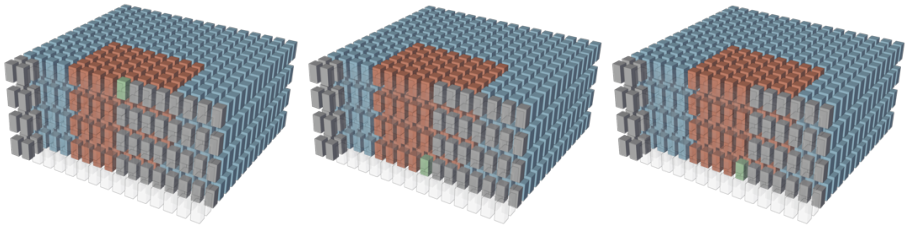
[1] Technical University of Munich, Munich, Germany
burakhan.koyuncu@tum.de
[2] Huawei Munich Research Center, Munich, Germany
[3] Huawei Moscow Research Center, Moscow, Russia
[4] Tencent America, Palo Alto, USA

A1 Illustration of the Runtime Optimization Methods

This section gives further details about the proposed runtime optimization methods; *skip intermediate channel segments* (SCS) and *batched dynamic sequence processing* (BDS). Fig. A1 illustrates a few of the processing steps of a Contextformer (cfo). Since the transformers are sequence-to-sequence models, processing step $n+3$ also calculates the output for step n. The provided mask allows using of subsequent elements for calculating the attention for each element. Thus, the calculation of the output for step n can be skipped during the encoding without harming the causality (SCS). In the figure, one can see that steps $n+3$ and $n+7$ contain equal number of latent elements used for context modeling, which enables processing them in batches efficiently (BDS). Our algorithm searches for latent elements, which can be grouped into a batch, and then models the context accordingly (see Alg. A1). Fig. A2 illustrates the wavefront processing for the decoding runtime optimization. Since latent tensor elements with the same processing order can be encoded and decoded independently, we process them in batches. The only requirement for using wavefront processing in the decoder is that the latent tensor elements are coded into the bitstream in the same order as in the encoder. That way, the entropy parameters can be computed regardless of the algorithm optimizing the encoding and ordered according to the wavefront processing index prior to bitstream coding.

A2 Architectural Details

Table A1 outlines the architectural details of the compression model with the Contextformer.



(a) Step n (b) Step n+3 (c) Step n+7

Fig. A1. Illustration of the context modeling in a Contextformer(cfo) with channel-first-order processing and sliding window attention for various processing steps n. The latent tensors are displayed in different colors: (■) the current latent elements to be coded; (■) the latent elements used by the context model; (■) previously coded elements; and (■) elements yet to be coded.

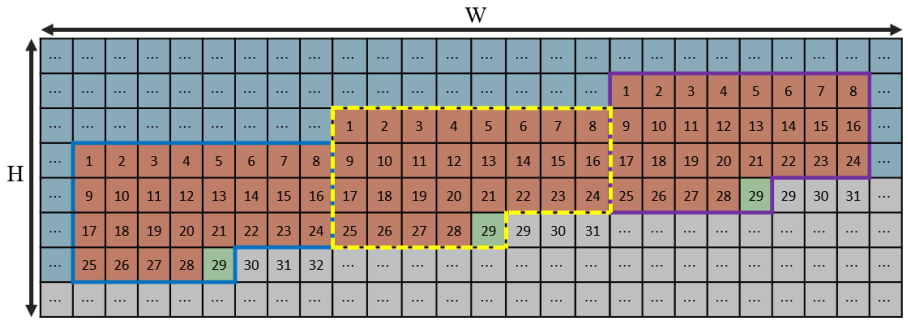


Fig. A2. Illustration of the context modeling in a Contextformer with sliding window attention and wavefront processing for an arbitrary processing step n. The processing order of each latent element is $n + N$, where N is the number given in the figure for each latent. Simultaneously processed windows for step $n + 29$ are framed in different colors. For simplicity, the channel dimension is omitted.

Algorithm A1: Runtime optimization for Contextformer

Input: Seq. Latent Tensor $\hat{\boldsymbol{y}}_{\boldsymbol{s}}$, Contextformer CTX
Output: Contextformer Output $\boldsymbol{\psi}$

$Q_{ctx}, Q_{crt} \leftarrow \{\}$ `// initialize empty dictionaries`
$\boldsymbol{\psi} \leftarrow \mathbf{0}$ `// initialize Context Model Output`
$H, W, N_{cs} \leftarrow get\_original\_shape(\hat{\boldsymbol{y}}_{\boldsymbol{s}})$
`//loop over sliding windows & store indices`
foreach *coord. (i,j,k)* ***in*** *3DMeshCube(H,W,N_{cs})* **do**
    `//get indices residing in current window`
    $indices \leftarrow get\_indices(i, j, k)$
    `//compute priority, e.g.` $length(idx)$ `for BDS`
    $priority \leftarrow get\_priority(i, j, k)$
    `//store current latent index`
    $Q_{crt}[priority].append((i, j, k))$
    `//store latent indices used as context`
    $Q_{ctx}[priority].append(indices)$
end
`//loop over coding priorities & run CTX`
foreach *priority* ***in*** *sorted(Q_{crr}.keys())* **do**
    `//restore latent indices`
    $i_{crt} \leftarrow Q_{crt}[priority]$
    $i_{ctx} \leftarrow Q_{ctx}[priority]$
    `//run context model for the current latents`
    $\hat{\boldsymbol{y}}_{ctx} \leftarrow \hat{\boldsymbol{y}}_{s}[i_{ctx}]$
    $\boldsymbol{\psi}[i_{crt}] \leftarrow CTX(\hat{\boldsymbol{y}}_{ctx})$
end

Table A1. An overview of the proposed model with the Contextformer, where each row depicts one layer or component of the model. "Conv: $K \times K \times N$ s2" stands for a convolutional layer with kernel size of $K \times K$, number of N output channels, and stride "s" of 2. Similarly, "Deconv" is an abbreviation for transposed convolutions. IGDN is the inverse function of GDN [4]. "Dense" layers are specified by their output dimension, whereas $K_1 = 2M + d_e$, $K_2 = (4k_m M)/N_{cs}$. We selected the base configuration of the Contextformer as $\{L = 8,\ d_e = 384,\ d_{mlp} = 4d_e,\ h = 12,\ N_{cs} = 4,\ co = cfo\}$

Encoder	Decoder	Hyperencoder	Hyperdecoder
Conv: 3×3×N s2	2×ResBlock: 3×3×M	Conv: 3×3×N s1	Conv: 3×3×M s1
GDN	Deconv: 3×3×N s2	Leaky ReLU	Deconv: 3×3×M s2
RNAB	IGDN	Conv: 3×3×N s1	Leaky ReLU
Conv: 3×3×N s2	Deconv: 3×3×N s2	Conv: 3×3×N s2	Conv: 3×3×M s1
GDN	IGDN	Leaky ReLU	Deconv: 3×3×$\frac{3}{2}M$ s2
Conv: 3×3×N s2	Deconv: 3×3×N s2	Conv: 3×3×N s1	Leaky ReLU
GDN	RNAB	Conv: 3×3×N s2	Deconv: 3×3×2M s2
Conv: 3×3×N s2	IGDN		Leaky ReLU
Conv: 1×1×M s2	Deconv: 3×3×3 s2		

Context Model	Entropy Parameters
Contextformer: $\{L, d_e, d_{mlp}, h, N_{cs}, co\}$	Dense: $(2K_1+K_2)/3$
	GELU
	Dense: $(K_1+2K_2)/3$
	GELU
	Dense: K_2

A3 Details of Experiments

A3.1 Obtaining Prior-Art Results

As explained in Section 5, we compare the performance of our model with various compression methods from the prior art. For this purpose, we used the official open-source implementations of those methods for the inference. For all methods, we used their default configurations. The links to those implementations are available in Table A2. Where the source is not available, we extracted the results from the corresponding publications. We also evaluated our extraction method by using it for the methods with available sources. We observed a negligible difference ($< 0.1\%$ in BD-Rate) between the inferred and extracted results.

A3.2 Detailed Performance Comparison

We provide more extensive versions of the figures presented in Section 5. Figures A3 and A4 show the same performance comparison on Kodak dataset as Figs. 3a and 4, but it includes additional prior-art methods such as Qian et al. [39] and Chen et al. [11]. Similarly, Figs. A5 and A7 illustrate the performance comparison on CLIC2020 and Tecnick datasets, which show the same results as Figs. 5a and 5b but provide better visuals. For the sake of completeness, we also provide the results on CLIC2020 and Tecnick datasets for our models optimized for MS-SSIM (see Figs. A6 and A8).

Table A2. Prior-art methods mentioned in this work with their official sources.

Method	Link
Minnen et al. [33] Minnen&Singh [34]	https://github.com/tensorflow/compression
Cheng et al. [12]	https://github.com/ZhengxueCheng/Learned-Image-Compression-with-GMM-and-Attention
Chen et al. [11]	https://github.com/NJUVISION/NIC
BPG [8]	https://bellard.org/bpg/
VTM 16.2 [43]	https://vcgit.hhi.fraunhofer.de/jvet/VVCSoftware_VTM

A3.3 Detailed Model Size Comparison

Table A3 shows the number of parameters of our compression framework compared to various frameworks. Depending on the implementation, we used the summary functions of PyTorch or Tensorflow for the calculation. Compared to the other transformer-based model such as Qian et al. [38], our model employs relatively complex encoder/decoder with a smaller bottleneck, and much simpler hyperprior. Additionally, our context model with spatio-channel attention requires an order of magnitude smaller parameters compared to the other context model adopting channelwise processing such as Minnen&Singh [34]. The total number of parameters in our entropy modeling is relatively smaller than the various approaches including our baseline Cui et al. [14]. Therefore, our method might have a higher potential for adopting online rate-distortion optimization techniques such as [51].

Table A3. Number of parameters of various models, which are calculated by summary functions of the used framework.

Method	Encoder	Decoder	Hyper Encoder	Hyper Decoder	Context+Entropy Parameters
Contextformer	8.1M	9.4M	1.6M	2.4M	15.9M
Cui et al. [14]	8.1M	9.4M	1.6M	2.4M	17.2M
Qian et al. [38]	3.8M	3.8M	8.2M	15.9M	13.1M
Minnen&Singh [34]	2.8M	2.8M	2.1M	2.1M	2.8M
Minnen et al. [33]	4.2M	4.2M	5.2M	5.8M	101.9M

A3.4 Visual Assessment

In order to assess qualitative performance, we visually compare the reconstructed images from our Contextformer models (base configuration), which are sep-

arately optimized for MSE and MS-SSIM, with the ones from BPG [8] and VTM 16.2 [43]. Fig. A9 shows the reconstructed images of *kodim23* (Kodak image dataset), and several crops from the images. Each reconstruction is generated by targetting approximately 0.06 bpp. Similarly, Fig. A10 shows the reconstructions of *kodim07* for 0.1 bpp. As we can see from both figures that the classical codecs suffer from artifacts such as smear, blocking, and aliasing, whereas our methods (both MSE and MS-SSIM trained models) preserve contours and high frequency details better, and provides higher PSNR and MS-SSIM performance for lower bpp. While both the MSE and MS-SSIM trained models provide better visual quality than classical codecs, the type of distortions introduced by the Contextformer are influenced by the cost function used. In our experiments, the model optimized for MSE is better at producing sharper edges, while the model optimized for MS-SSIM is better at preserving structure and texture of the objects.

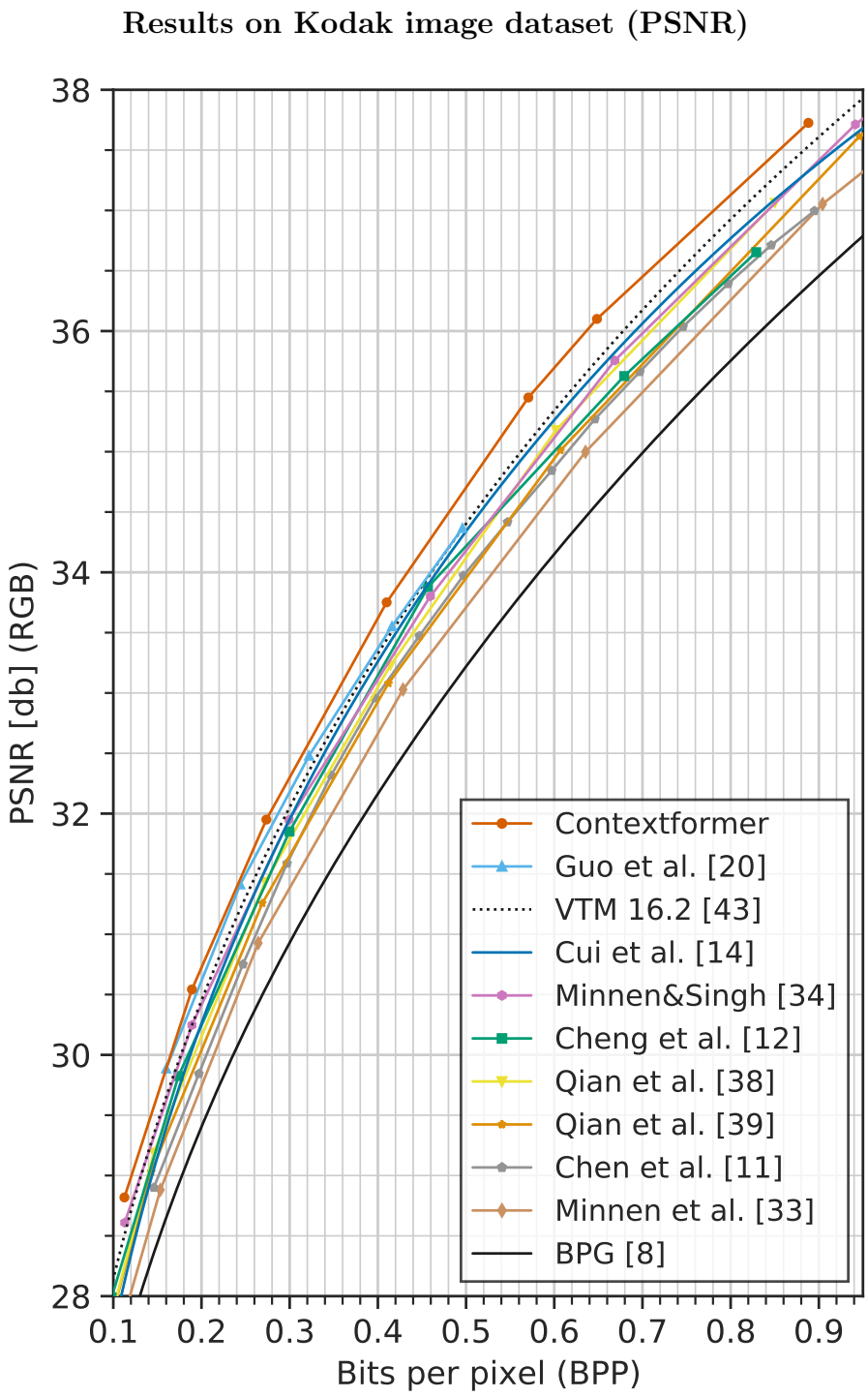


Fig. A3. 347.12354ptRate-distortion performance comparison on Kodak dataset in terms of PSNR for our model, and various learning-based and classical codecs.

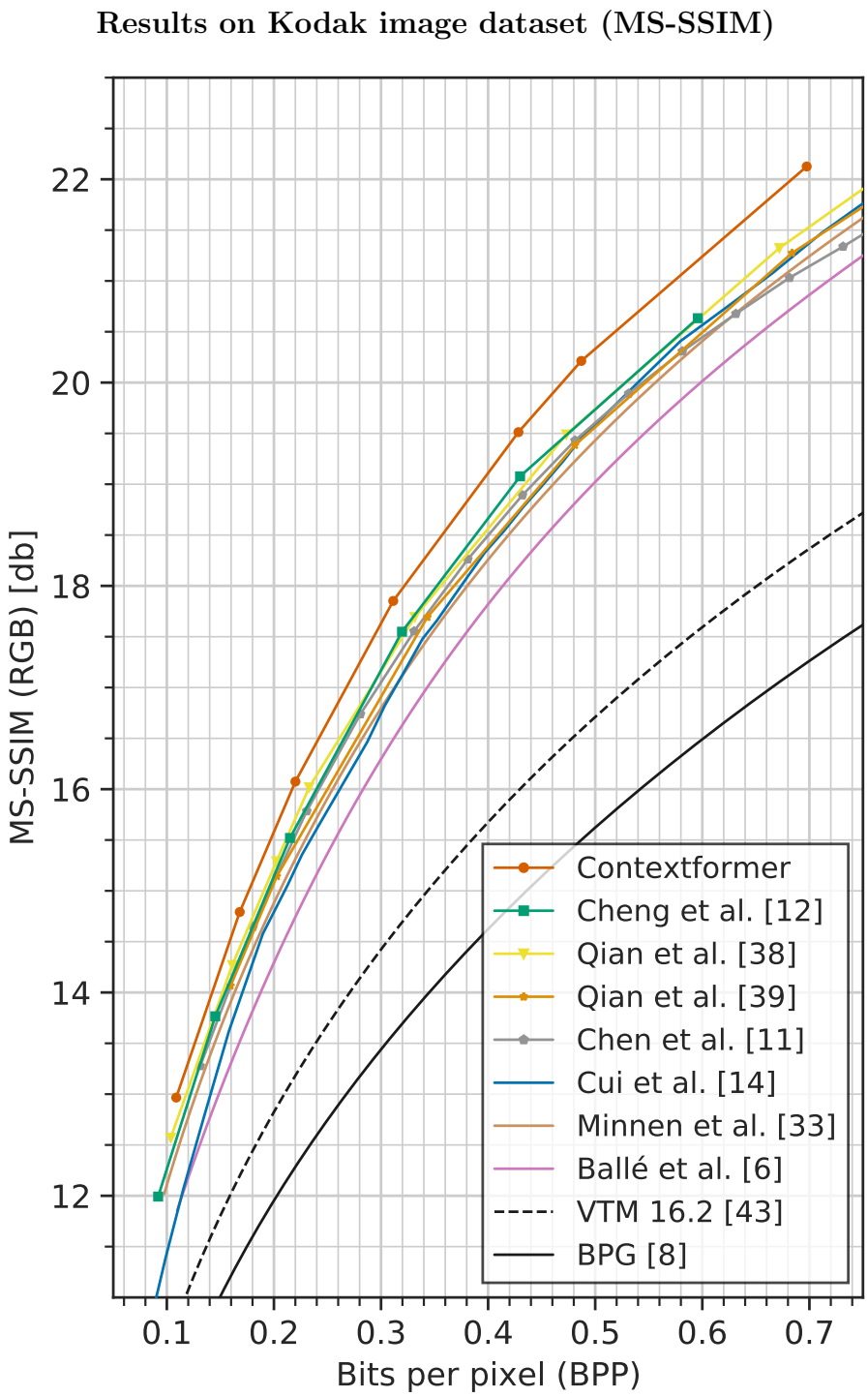


Fig. A4. Rate-distortion performance comparison on Kodak dataset in terms of MS-SSIM for our model, and various learning-based and classical codecs.

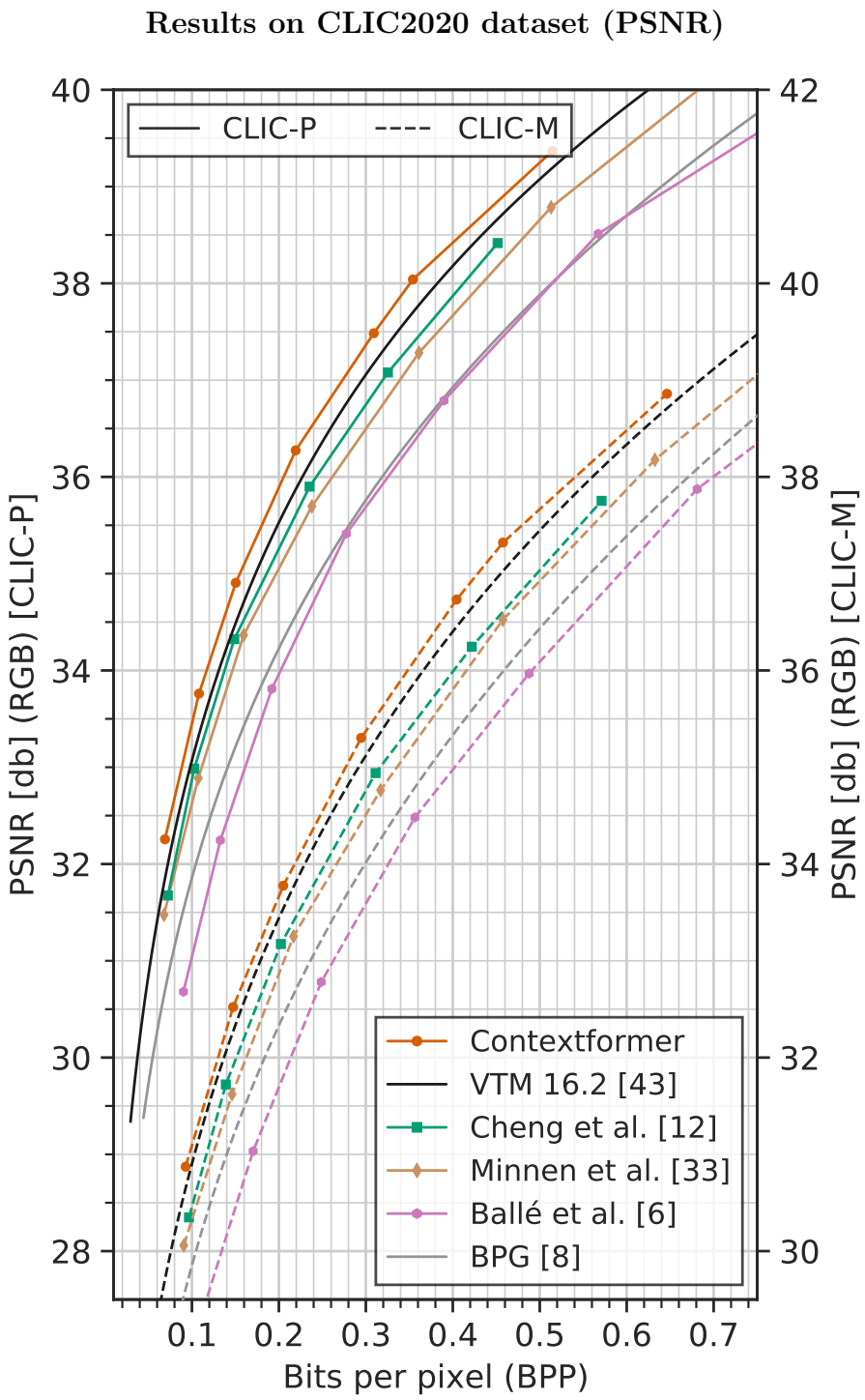


Fig. A5. Rate-distortion performance comparison on CLIC-Professional (solid line, left vertical axis) and CLIC-Mobile (dashed line, right vertical axis) datasets in terms of PSNR for our model, and various learning-based and classical codecs.

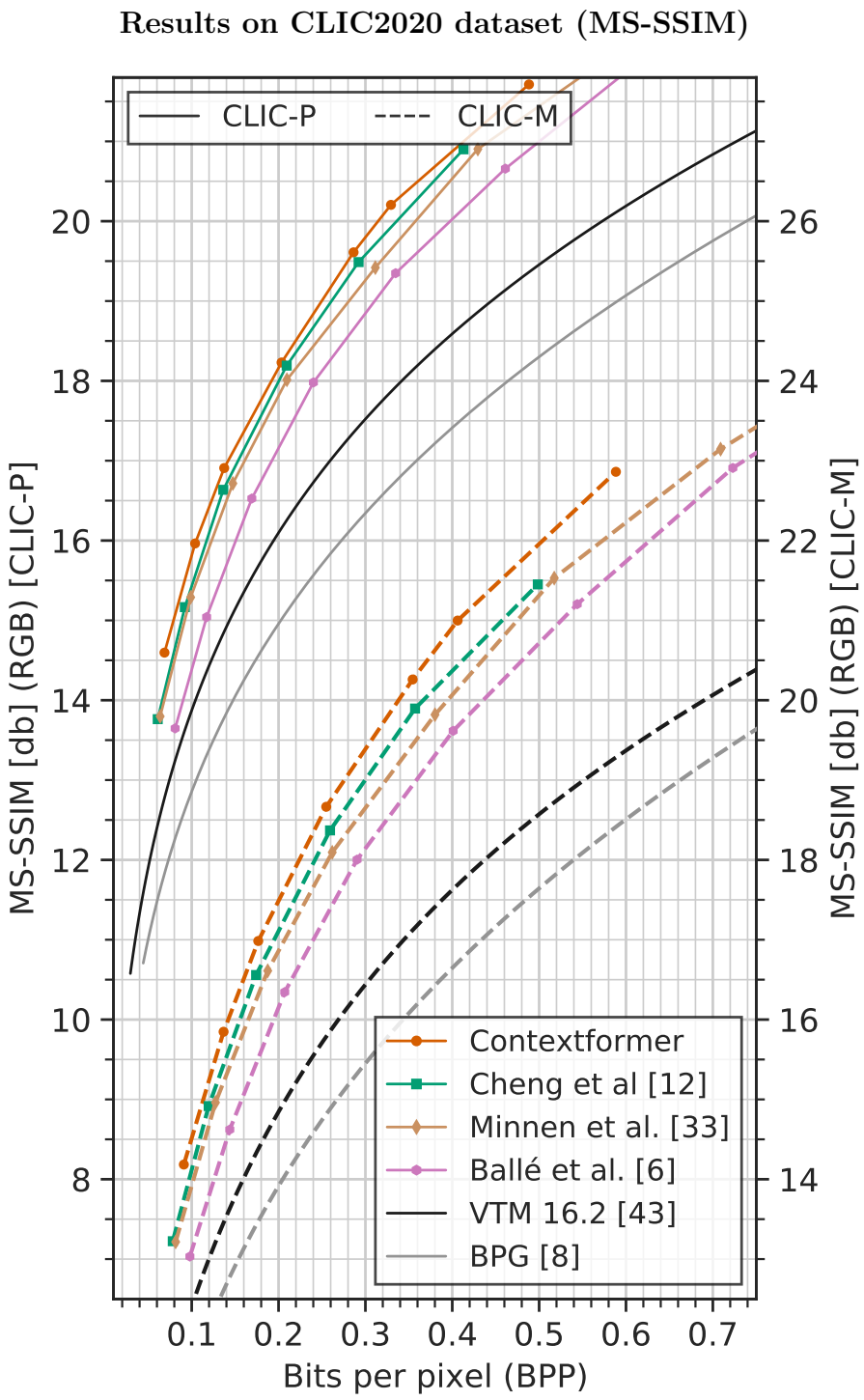


Fig. A6. Rate-distortion performance comparison on CLIC-Professional (solid line, left vertical axis) and CLIC-Mobile (dashed line, right vertical axis) datasets in terms of MS-SSIM for our model, and various learning-based and classical codecs.

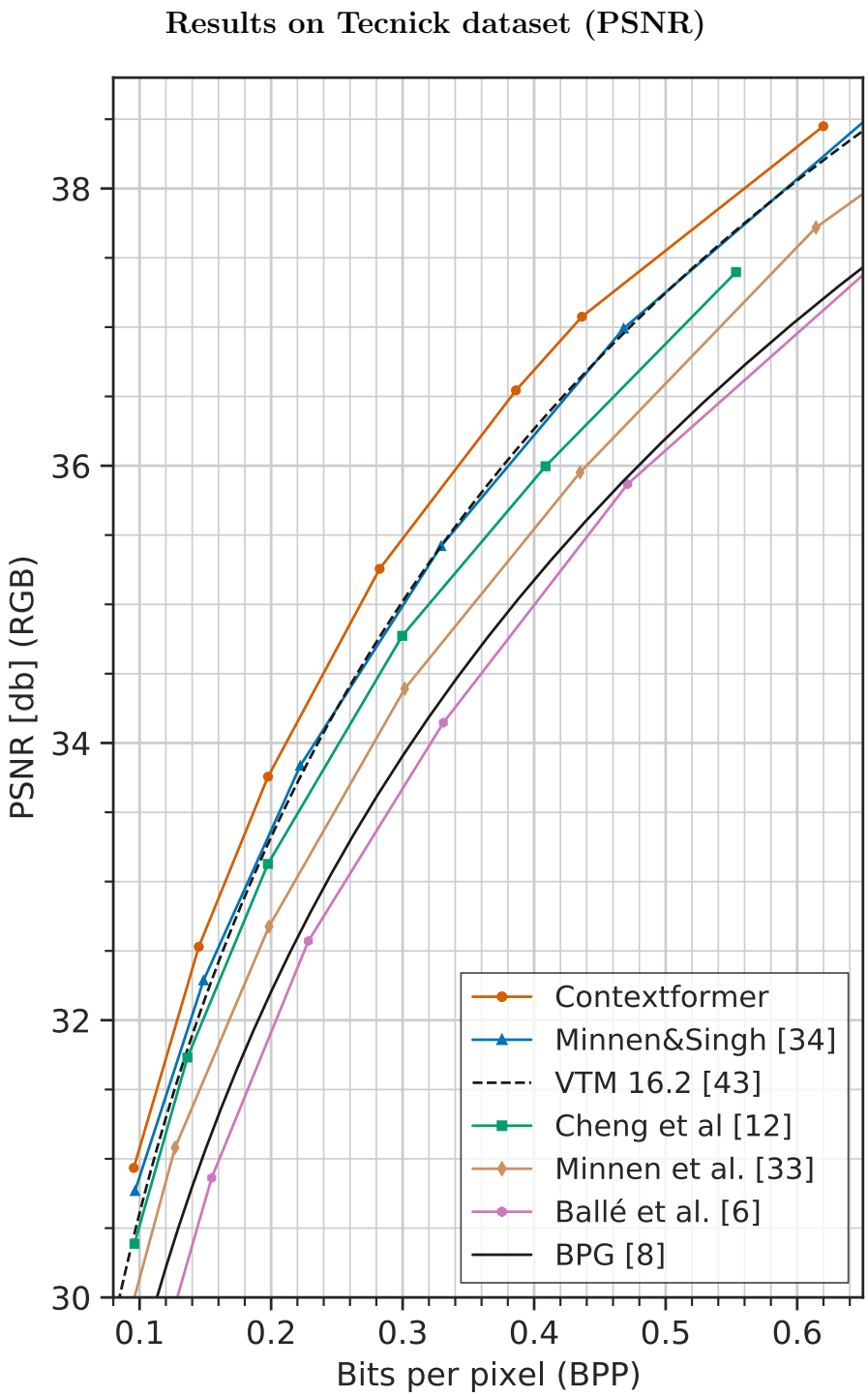


Fig. A7. Rate-distortion performance comparison on Tecnick dataset in terms of PSNR for our model, and various learning-based and classical codecs.

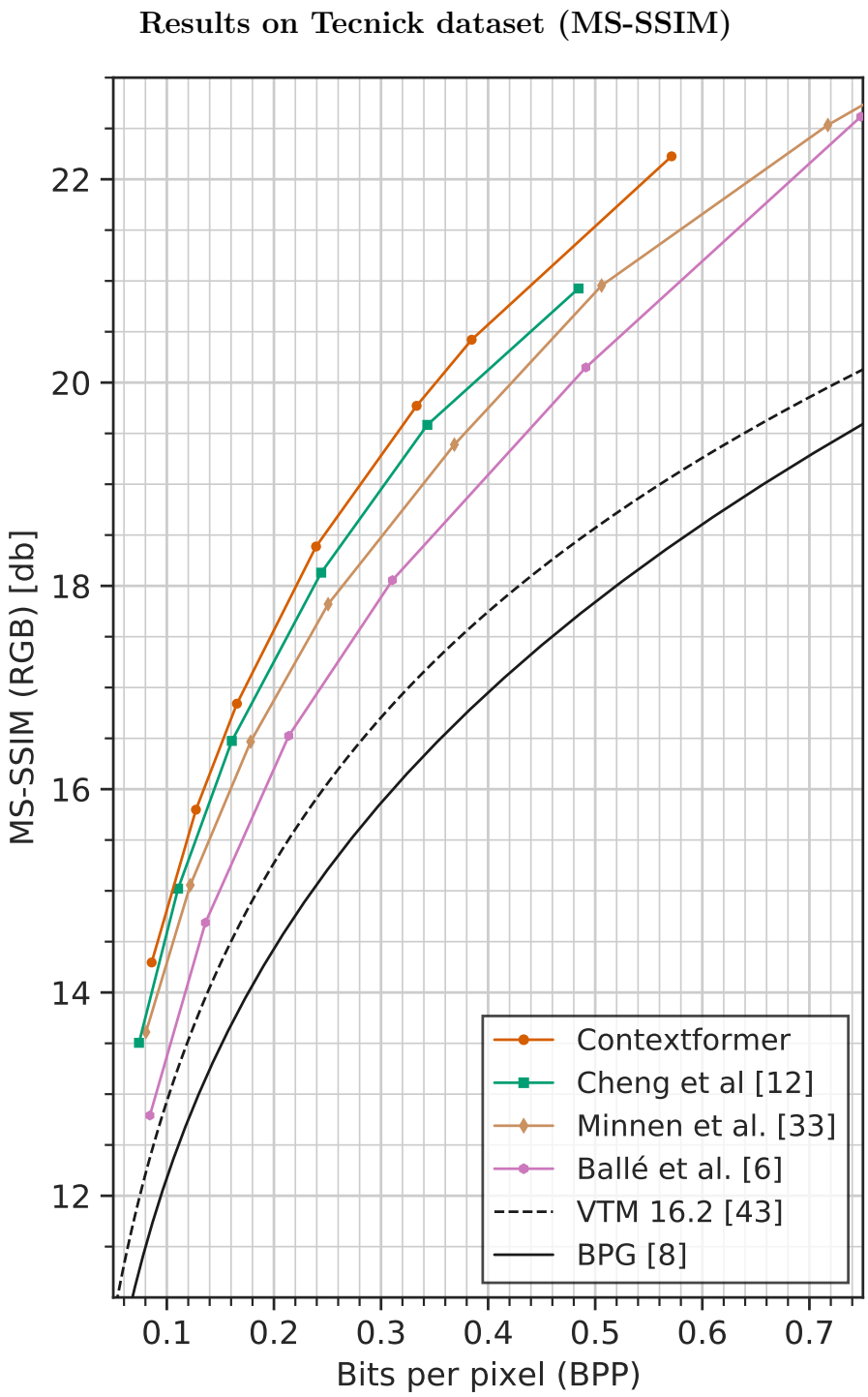


Fig. A8. Rate-distortion performance comparison on Tecnick dataset in terms of MS-SSIM for our model, and various learning-based and classical codecs.

Fig. A9. The reconstructed images of *kodim23* from the Kodak image dataset for the visual comparison. The image is compressed by our models (optimized for MSE or MS-SSIM), VTM 16.2, and BPG for the target bpp of 0.06.

Fig. A10. The reconstructed images of *kodim07* from the Kodak image dataset for the visual comparison. The image is compressed by our models (optimized for MSE or MS-SSIM), VTM 16.2, and BPG for the target bpp of 0.1.