Learned Image Compression with Hierarchical Progressive Context Modeling

Supplementary Material

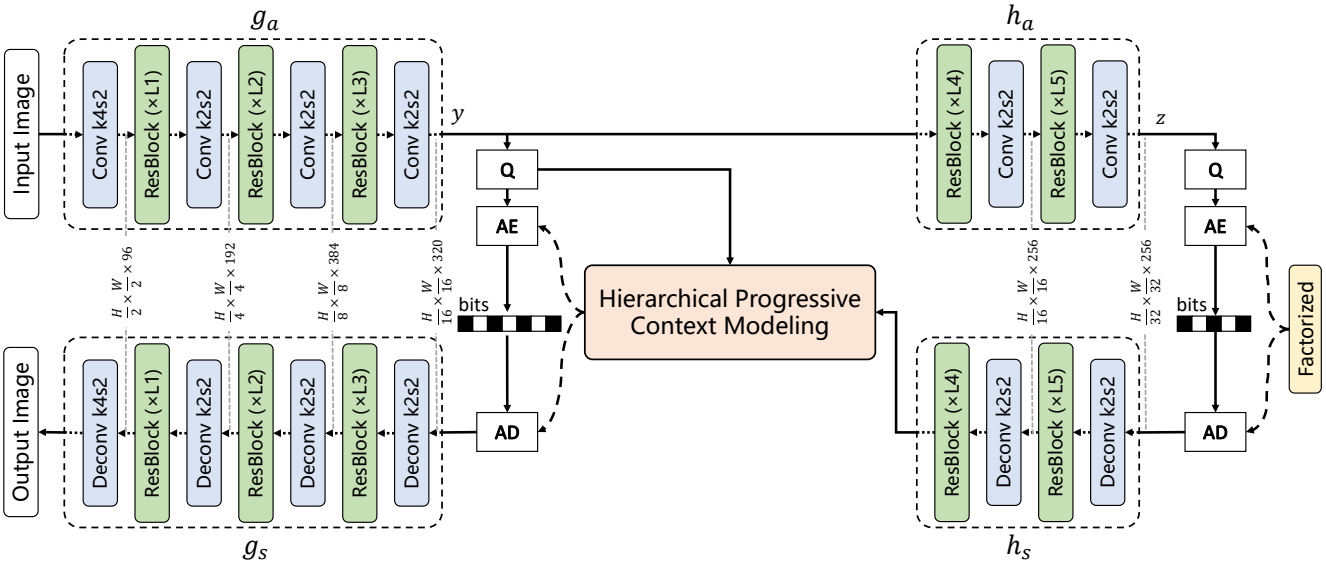


Figure A. Overall architecture of our proposed model. The detailed structure of the ResBlock is shown in Fig. D. 'k2s2' represents the convolution layer with kernel size as 2 and stride as 2.

A. Overall architecture

The overall architecture of our proposed model is shown in Fig. A. Following the design of transform networks in previous learned image compression methods [4], we adopt residual blocks along with down-sampling and up-sampling layers to establish nonlinear transforms g_a, g_s, h_a and h_s. Specifically, we use a single convolution layer with a stride of 2 or 4 for both down-sampling and up-sampling operations. The kernel size of the input down-sampling layer and output up-sampling layer is set as 4, and the kernel sizes in other down-sampling layers and up-sampling layers are set as 2 to reduce complexity. Additionally, we utilize the advanced FasterNet [2] blocks as the ResBlock in our model. The detailed structure of the ResBlock is presented in Fig. D (a). The PConv is the partial convolution layer, which processes spatial dense convolution only on partial channels. In our HPCM-Base model, the number of ResBlocks at different stages is set as $[L1, L2, L3, L4, L5] = [2, 2, 4, 1, 3]$. In our HPCM-Large model, the depth of $L3$ stage is increased for enhanced transform capacity, $[L1, L2, L3, L4, L5] = [2, 2, 8, 1, 3]$.

B. Structure of Entropy Parameter Networks

Figure C shows the network structure of the entropy parameter network g_{ep}. g_{ep} contains a 1×1 convolution layer followed by multiple DepthConvBlocks. The detailed structure of the DepthConvBlock is shown in Fig.D (b). To reduce the model parameters, we share the weights of DepthConvBlocks across different coding steps. We use two different entropy parameter networks. One is used for coding $\hat{y}^{S1}$ and $\hat{y}^{S2}$, denoted as g_{ep}^{S1+S2}. Another is used for coding $\hat{y}^{S3}$, denoted as g_{ep}^{S3}. To enhance the adaptability of shared networks, we introduce step adaptive embedding into entropy parameter networks to modulate the weight of each channel. In our HPCM-Base model, the numbers of DepthConvBlocks are set as $[N1, N2] = [2, 1]$ and $[N1, N2] = [3, 2]$ in g_{ep}^{S1+S2} and g_{ep}^{S3}, respectively. In our HPCM-Large model, the numbers of DepthConvBlocks are set as $[N1, N2] = [2, 2]$ and $[N1, N2] = [4, 3]$ in g_{ep}^{S1+S2} and g_{ep}^{S3}, respectively.

C. Detailed Hierarchical Coding Schedule

Figure E presents the detailed multi-scale partition process and coding schedules on eight channel groups of $\hat{y}$. Different multi-scale partition methods are applied to different channel groups to enable the interaction of spatial and channel-wise context information. We design the coding schedule on $\hat{y}^{S3}$ following previous studies [7, 9], which fully exploits the spatial correlation and enhances context diversity.

Figure F illustrates the detailed coding process for different coding step allocations. We present the coding step

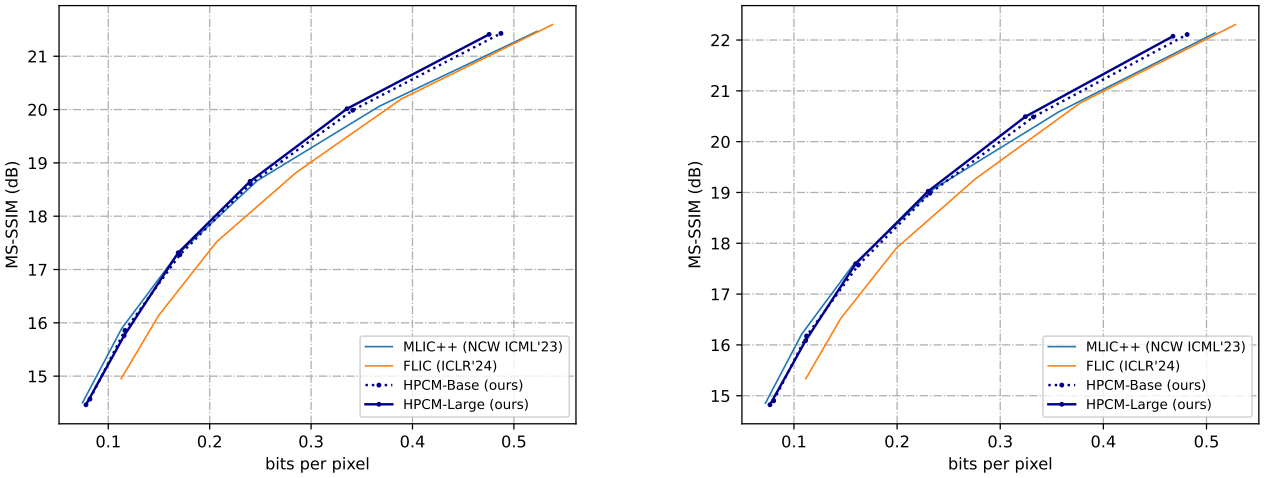


Figure B. Rate-distortion curves on different datasets. The left one is tested on the CLIC Pro Valid dataset, and the right one is tested on the Tecnick dataset.

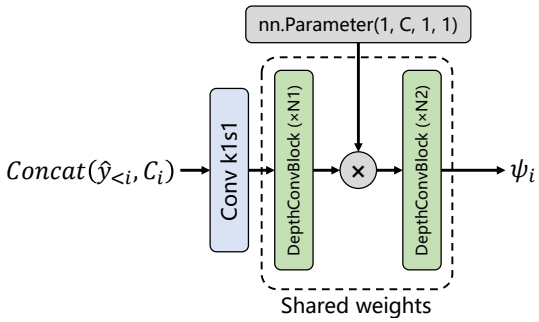


Figure C. The network structure of entropy parameter network g_{ep}. The detailed structure of the DepthConvBlock is illustrated in Fig. D.

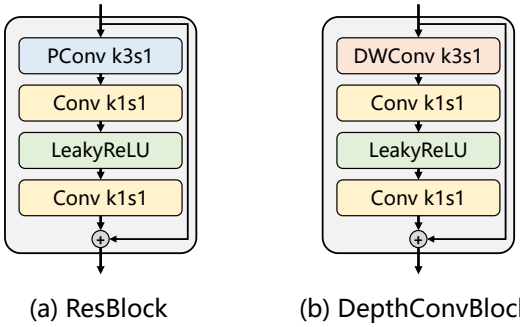


Figure D. The detailed structure of the ResBlock and DepthConvBlock.

(2, 3, 3), (2, 3, 12), and (4, 3, 6) for coding $(\hat{y}^{S1}, \hat{y}^{S2}, \hat{y}^{S3})$, respectively.

Table A. Comparison of training speed of various methods.

Model	Training Speed ↑ (steps/s)
ELIC (CVPR'22) [4]	4.07
STF (CVPR'22) [11]	3.31
TCM (CVPR'23) [8]	1.28
MLIC++ (NCW ICML'23) [5]	1.24
FLIC (ICLR'24) [6]	1.93
WeConvene (ECCV'24) [3]	1.43
HPCM-Base (ours)	3.67
HPCM-Large (ours)	2.88

D. Comparison of Training Speed

We compare the training speed of the proposed HPCM-Base and HPCM-Large with recent advanced methods in Table A. The training time is evaluated on one NVIDIA GeForce RTX 3090 GPU, with training batch size as 8 images and patch size as 256×256. Compared to recent state-of-the-art learned image compression methods like [5, 6], our methods achieve faster training speed.

E. Additional Rate-Distortion Performance

Figure B presents the rate-distortion performance of our model optimized for MS-SSIM. Compared to recent advanced learned image compression methods, our proposed methods achieve superior performance at higher bitrates.

F. Additional Results on Coding Time

Table C breaks down coding times into network inference (T_{Net}) and arithmetic coding (T_{AC}). For prior studies, we used their open-sourced models and code, as shown in Ta-

Table B. Code links of various methods. We use these open-source implementations to evaluate the compression performance and computational complexity of each model.

Model	Code Link
ELIC (CVPR'22) [4]	https://github.com/JiangWeibeta/ELIC
STF (CVPR'22) [11]	https://github.com/Googolxx/STF
TCM (CVPR'23) [8]	https://github.com/jmliu206/LIC_TCM
MLIC++ (NCW ICML'23) [5]	https://github.com/JiangWeibeta/ELIC
FLIC (ICLR'24) [6]	https://github.com/qingshi9974/ICLR2024-FTIC
MambaVC (Arxiv'24) [10]	https://github.com/QinSY123/2024-MambaVC
WeConvene (ECCV'24) [3]	https://github.com/fengyurenpingsheng/WeConvene

Table C. Coding times of various methods. Times are in milliseconds (ms).

Models	Encoding			Decoding			kMACs/pixel
	T_{Net}	T_{AC}	T_{Total}	T_{Net}	T_{AC}	T_{Total}	
ELIC	44	82	126	37	74	111	573.88
TCM	89	110	199	83	119	202	1823.58
MLIC++	73	117	190	74	152	226	1282.81
HPCM-Base	58	25	83	57	24	81	918.57

Table D. Abaltion studies on the number of hierarchical stages.

Models	kMACs/pixel	BD-Rate
w/o hierarchical (1-stage)	1107.48	1.07%
2-stage	954.17	0.35%
HPCM-Base (3-stage)	918.57	**0.00%**
4-stage	**911.44**	0.18%

ble B. Across different models, T_{Net} generally increases with higher kMACs/pixel; this aligns with HPCM-Base exhibiting a higher T_{Net} than ELIC. As for T_{AC}, it varies across models due to different implementations in the released code. Our arithmetic coding implementation improves upon the widely used CompressAI-based [1] implementation by enabling more efficient data exchange between Python and C. This optimization significantly reduces T_{AC}.

Since coding time is implementation-dependent, we focus on comparing the kMACs/pixel to measure computational complexity. Fig. **??** in the maintext plots "kMACs/pixel vs. BD-Rate", showing that our method achieves superior performance-complexity trade-offs compared to current SOTA methods.

G. Additional Ablation Studies

G.1. Ablation Studies on Hierarchical Coding Stages

We further provide ablation studies to verify the superiority of the 3-stage model. As shown in Table D, the 3-stage model achieves lower kMACs/pixel and better performance compared to the 1-stage and 2-stage variants, as it more effectively captures long-range spatial contexts. Increasing the number of stages to 4 slightly reduces computational complexity, but the performance is marginally worse. This is because allocating coding steps to capture extremely long-range spatial contexts is not cost-effective. Therefore, we adopt the 3-stage model as HPCM-base.

Table E. Ablation studies on shared parameters in context models.

Models	kMACs/pixel	Params (M)	BD-Rate
HPCM-Base	918.57	68.50	0.00%
w/o shared params	918.57	189.99	-0.15%

G.2. Ablation Studies on Shared Parameters of Context Models

We have tested a variant of HPCM-Base using context models with non-shared parameters at **all** scales. As shown in Table E, this model achieves comparable performance but with a substantial increase in the number of parameters. This indicates that employing different network settings across scales offers limited benefits to our model. Therefore, we shared the parameters of context models at all scales to significantly reduce the model's parameter count.

H. Additional Visual Comparison Results

Fig. G and Fig. H presents the reconstructed images of our proposed HPCM-Base, HPCM-Large, and various methods [4, 5, 11].

References

[1] Jean Bégaint, Fabien Racapé, Simon Feltman, and Akshay Pushparaja. CompressAI: A PyTorch library and evaluation platform for end-to-end compression research. *arXiv preprint arXiv:2011.03029*, 2020. 3

[2] Jierun Chen, Shiu-hong Kao, Hao He, Weipeng Zhuo, Song Wen, Chul-Ho Lee, and S.-H. Gary Chan. Run, don't walk: Chasing higher FLOPS for faster neural networks. In *IEEE/CVF Conference on Computer Vi-*

Figure E. The detailed coding progress of our proposed hierarchical coding schedule.

sion and Pattern Recognition (CVPR), pages 12021–12031, 2023. 1

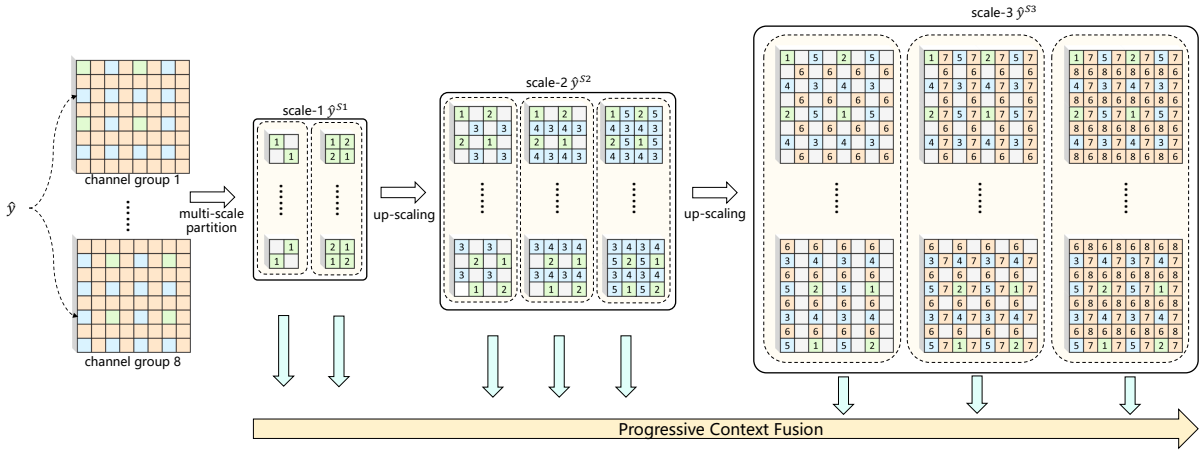
[3] Haisheng Fu, Jie Liang, Zhenman Fang, Jingning Han, Feng Liang, and Guohe Zhang. Weconvene: Learned image compression with wavelet-domain convolution and entropy model. In *European Conference on Computer Vision (ECCV)*, pages 37–53, 2024. 2, 3

[4] 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 *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5708–5717, 2022. 1, 2, 3

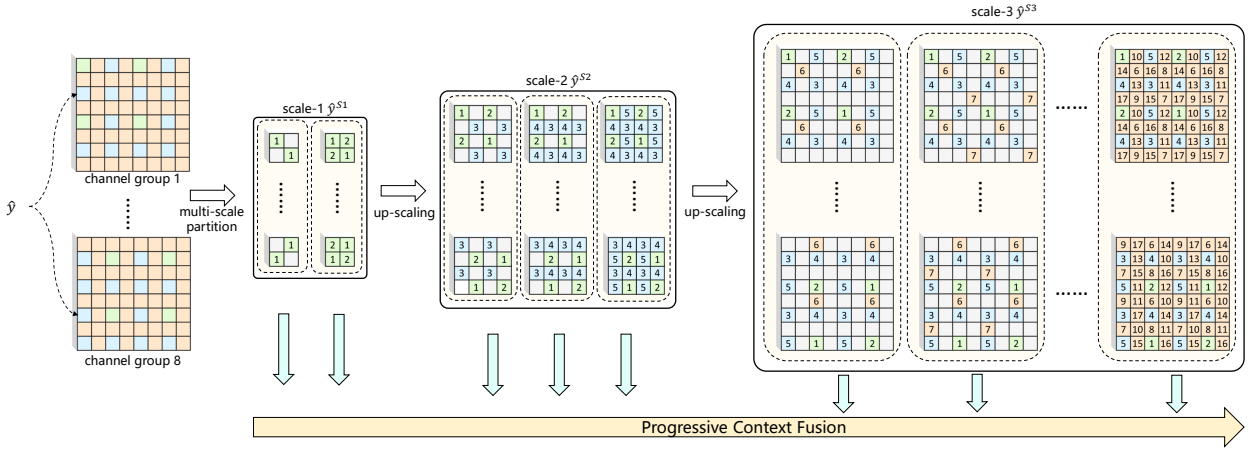
[5] Wei Jiang and Ronggang Wang. MLIC++: Linear complexity multi-reference entropy modeling for learned image compression. In *ICML Workshop*, 2023. 2, 3

[6] Han Li, Shaohui Li, Wenrui Dai, Chenglin Li, Junni Zou, and Hongkai Xiong. Frequency-aware transformer for learned image compression. In *International Conference on Learning Representations (ICLR)*, 2024. 2, 3

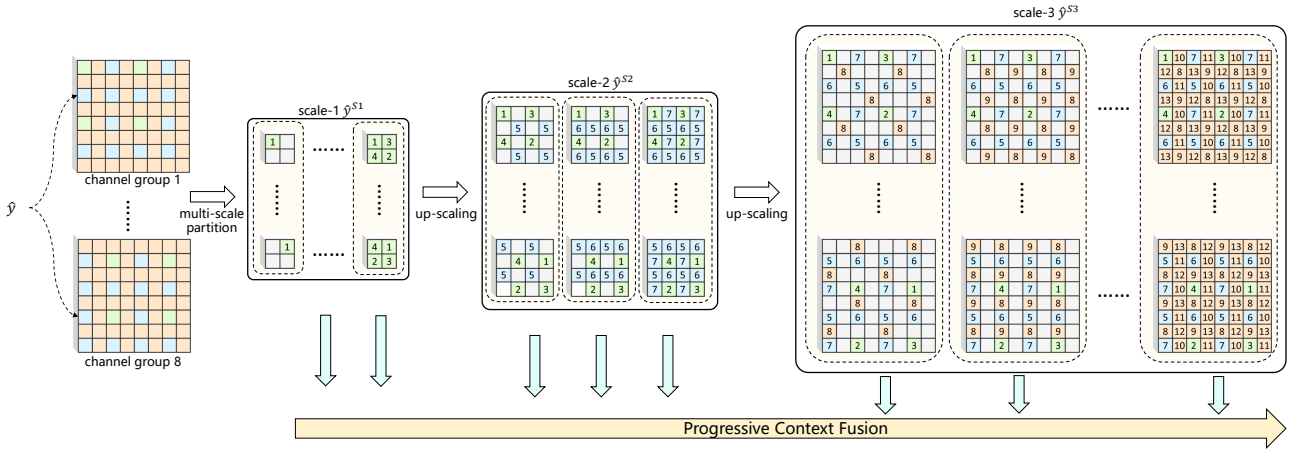
[7] Yuqi Li, Haotian Zhang, and Dong Liu. Flexible coding order for learned image compression. In *IEEE International Conference on Visual Communications and Image Processing (VCIP)*, pages 1–5, 2023. 1



(a) Coding step (2, 3, 3) on scale ($\hat{y}^{S1}, \hat{y}^{S2}, \hat{y}^{S3}$). There are 8 coding steps in total.



(b) Coding step (2, 3, 12) on scale ($\hat{y}^{S1}, \hat{y}^{S2}, \hat{y}^{S3}$). There are 17 coding steps in total.



(c) Coding step (4, 3, 6) on scale ($\hat{y}^{S1}, \hat{y}^{S2}, \hat{y}^{S3}$). There are 13 coding steps in total.

Figure F. The detailed coding process for (a) coding step (2, 3, 3), (b) coding step (2, 3, 12), and (c) coding step (4, 3, 6) for coding ($\hat{y}^{S1}, \hat{y}^{S2}, \hat{y}^{S3}$).

Figure G. Visual comparison of reconstructed images of Kodim04 in the Kodak dataset with various learned image compression methods.

[8] Jinming Liu, Heming Sun, and Jiro Katto. Learned image compression with mixed Transformer-CNN architectures. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14388–14397, 2023. 2, 3

[9] Fabian Mentzer, Eirikur Agustson, and Michael Tschannen. M2T: Masking Transformers twice for faster decoding. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 5317–5326, 2023. 1

[10] Shiyu Qin, Jinpeng Wang, Yiming Zhou, Bin Chen, Tianci Luo, Baoyi An, Tao Dai, Shutao Xia, and Yaowei Wang. Mambavc: Learned visual compression with selective state spaces. *arXiv preprint arXiv:2405.15413*, 2024. 3

[11] Renjie Zou, Chunfeng Song, and Zhaoxiang Zhang. The devil is in the details: Window-based attention for image compression. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 17471–17480, 2022. 2, 3

Original: bpp / PSNR

HCPM-Base (ours): 0.2010 / 31.93

HCPM-Large (ours): 0.1677 / 31.59

ELIC: 0.2369 / 32.03

STF: 0.2501 / 32.06

MLIC: 0.2125 / 32.14

Figure H. Visual comparison of reconstructed images of Kodim19 in the Kodak dataset with various learned image compression methods.