A Notation List

Table 3: Symbols used in this paper.

$\mathbf{x}$	Data/Observation node.
$\mathbf{y}$	Latent node.
$\mathbf{z}$	Hyper-latent node.
$\hat{\mathbf{y}}$	Samples of $\mathbf{y}$. Note that all symbols with a hat stands for samples.
$\mathbf{X}$	Data/Observation node set.
$\mathbf{L}$	Latent node set.
$\mathbf{G}$	BayesNet structure node set.
$f(\mathbf{G}), f(\mathbf{G}, \mathbf{L})$	Optimization target function.
$\mathcal{L}_R, \mathcal{L}_D, \mathcal{L}_C$	Loss function for rate, distortion and complexity.
λ_D, λ_C	Hyperparameters to adjust trade-off between rate, distortion and complexity.
$\mathbf{G}_{inter}$	Inter-node BayesNet structure node set.
$\mathbf{G}_{intra}$	Intra-node BayesNet structure node set.
$\mathbf{G}_{inter}^{\mathbf{y},\mathbf{z}}$	Inter-node BayesNet structure node for $p(\mathbf{y}\|\mathbf{z})$ (or $q(\mathbf{y}\|\mathbf{z})$).
π	Logit parameters for Bernoulli/Categorical distribution.
$\mathrm{T}_{c,h,w}$	Multipartite graph topological index node on channel dimension c, height dimension h and width dimension w.
$\mathbf{G}_{intra}^{i}$	The i-th Monte-Carlo sample of $\mathbf{G}_{intra}$
$g(\mathbf{N})$	Unconditional generative model (for intra-node BayesNet)
$W_{c,h,w}$	Dynamic convolution kernel weight to convolute with the input on channel dimension c, height dimension h and width dimension w.
g_a, g_s, h_a, h_s	Analysis, synthesis, hyper-analysis, hyper-synthesis models, respectively. Modules defined in [4].

B Further Implementation Details

B.1 Intra-node BayesNet Implementation

Computational Complexity of Multipartite-based Dynamic Masked Convolution Building on the technique outlined in Section 4.2, Equation 16 and Equation 17 provide an instantiation of intra-node BayesNet via multipartite-based dynamic masked convolution. Here we further provide a PyTorch[6]-style pseudo-code implementation in Algorithm 1 to clarify the implementation details. Measuring with MAC, the complexity of the multipartite-based dynamic masked convolution could be calculated by counting the MACs in convolution operations by:

$$\begin{aligned} \mathrm{MAC}_{\mathrm{dynconv}} &= C \times H_{out} \times W_{out} \times ((C_{out}//C) \times (C_{in}//C) \times C \times K \times K) \\ &= H_{out} \times W_{out} \times C_{out} \times C_{in} \times K \times K \end{aligned} \tag{20}$$

[6] `https://pytorch.org/`

We could see that the proposed operator has equivalent parameter count and computational complexity to that of standard 2D convolution operators. In practice, however, it usually runs slower on GPUs compared to traditional 2D convolutions, which is partly attributable to our current implementation not being fully GPU-optimized.

Algorithm 1 Pseudocode for Multipartite-based Dynamic Masked Convolution.

Require: Kernel size K, Output channels C_{out}, Input channels C_{in}, Dynamic channel groups (or channels of topological indices) C, Input tensor $\hat{\mathbf{y}} \in \mathbb{R}^{C_{in} \times H_{in} \times W_{in}}$, Topological indices $\hat{\boldsymbol{T}} \in \mathbb{R}^{C \times H_{in} \times W_{in}}$, Kernel weight $\mathsf{W} \in \mathbb{R}^{C_{out} \times C_{in} \times K \times K}$, Kernel bias $\mathsf{B} \in \mathbb{R}^{C_{out}}$,

Ensure: Output tensor $\hat{\mathbf{y}}_{out} \in \mathbb{R}^{C_{out} \times H_{out} \times W_{out}}$

Reshape $\hat{\mathbf{y}} \in \mathbb{R}^{(C_{in}//C) \times C \times H_{in} \times W_{in}}$

Reshape $\mathsf{W} \in \mathbb{R}^{C \times (C_{out}//C) \times (C_{in}//C) \times C \times K \times K}$

Reshape $\mathsf{B} \in \mathbb{R}^{C \times (C_{out}//C)}$

Initialize $\hat{\mathbf{y}}_{out} \in \mathbb{R}^{(C_{out}//C) \times C \times H_{out} \times W_{out}}$

for c, h, w in range(C, H, W) **do**

  Unfold2D $\hat{\mathbf{y}}_{knl} \in \mathbb{R}^{(C_{in}//C) \times C \times K \times K}$

  Unfold2D $\hat{T}_{knl} \in \mathbb{R}^{1 \times C \times K \times K}$

  Get mask $\hat{\mathbf{y}}_{mask} \leftarrow Binarize(\hat{T}_{knl} > \hat{T}[c, h, w])$ ▷ $\hat{\mathbf{y}}_{mask} \in \mathbb{R}^{1 \times C \times K \times K}$

  Apply mask $\hat{\mathbf{y}}_{input} \leftarrow (\hat{\mathbf{y}}_{mask} \cdot \hat{\mathbf{y}}_{knl})$ ▷ $\hat{\mathbf{y}}_{input} \in \mathbb{R}^{(C_{in}//C) \times C \times K \times K}$

  Convolution $\hat{\mathbf{y}}_{conv} \leftarrow \sum^{(C_{in}//C), C, K, K} (\hat{\mathbf{y}}_{input} \cdot \mathsf{W}[c]) + \mathsf{B}[c]$ ▷ $\hat{\mathbf{y}}_{conv} \in \mathbb{R}^{(C_{out}//C)}$

  Output $\hat{\mathbf{y}}_{out}[:, c, h, w] \leftarrow \hat{\mathbf{y}}_{conv}$

end for

Reshape $\hat{\mathbf{y}}_{out} \in \mathbb{R}^{C_{out} \times H \times W}$

Merging Predictions from Intra-node and Inter-node BayesNet In the original framework, the complete prior for predicting $\hat{\mathbf{y}}$ given $\hat{\mathbf{z}}$, denoted as $p(\hat{\mathbf{y}}|\hat{\mathbf{z}})$, fuses the output from $h_s(\hat{\mathbf{z}})$ and the context model $\mathrm{AR}(\hat{\mathbf{y}})$. This is achieved using a 3-layer MLP, consisting of 1×1 convolutional layers and LeakyReLU activation, as follows:

$$p(\hat{\mathbf{y}}|\hat{\mathbf{z}}) \propto \mathrm{MLP}(\mathrm{Concat}(h_s(\hat{\mathbf{z}}), \mathrm{AR}(\hat{\mathbf{y}}))). \tag{21}$$

In our approach to formulating the complete prior, $p(\hat{\mathbf{y}}|\hat{\mathbf{z}}, \mathbf{G}^{\hat{\mathbf{y}},\hat{\mathbf{z}}}_{inter}, \mathbf{G}^{\hat{\mathbf{y}}}_{intra})$, a similar 3-layer network is applied for merging. However, the nature of our autoregressive process across the channel dimension precludes the use of standard 1×1 convolutions. Instead, we employ the dynamic masked convolution with 1×1 kernels, accommodating autoregressive operations over three dimensions. Importantly, the output from $h_s(\hat{\mathbf{z}}, \mathbf{G}^{\hat{\mathbf{y}},\hat{\mathbf{z}}}_{inter})$ is treated as the initial partite, preceding the context model's output $\mathrm{AR}(\hat{\mathbf{y}}, \mathbf{G}^{\hat{\mathbf{y}}}_{intra})$ because $\hat{\mathbf{z}}$ has topological precedence

over $\hat{\mathbf{y}}$ in the BayesNet structure. This is represented as:

$$\begin{aligned} &p(\hat{\mathbf{y}}|\hat{\mathbf{z}}, \mathbf{G}_{inter}^{\hat{\mathbf{y}},\hat{\mathbf{z}}}, \mathbf{G}_{intra}^{\hat{\mathbf{y}}}) \propto \\ &\text{MLP}_{dyn}(\text{Concat}(h_s(\hat{\mathbf{z}}, \mathbf{G}_{inter}^{\hat{\mathbf{y}},\hat{\mathbf{z}}}), \text{AR}(\hat{\mathbf{y}}, \mathbf{G}_{intra}^{\hat{\mathbf{y}}})), \text{Concat}(\hat{T}^{\hat{\mathbf{y}}}, \hat{T}^{\hat{\mathbf{z}}})), \\ &\hat{T}^{\hat{\mathbf{y}}} \in \{0, 1, \ldots, S_{intra}\}, \quad \hat{T}^{\hat{\mathbf{z}}} \in -\mathbf{1}. \end{aligned} \tag{22}$$

Our merging neural network, when compared to [25], maintains the same MACs and parameters for equivalent channel configurations, as per Equation 20. Additionally, when contrasted with channel-wise autoregressive models [26], which employ several merging networks for channel groups, our method is also more MAC and parameter efficient, as the h_s prediction is processed solely once.

Learning the Intra-node BayesNet Generator Earlier in Section 4.2, we outlined the use of a generative model $g(\mathbf{N})$ in Equation 15 to model the latent correlations among nodes in the intra-node BayesNet. This model is a straightforward 2-layer network comprising Linear and ReLU layers. Nonetheless, we encountered optimization challenges leading to local optima that resulted in ineffective BayesNets. This is often due to the most effective intra-node BayesNets having diverse topological indices for adjacent nodes, which increases the divergence of their corresponding distributions and causes the Monte-Carlo sampler to converge prematurely.

To address this, we employed two strategies in optimizing the generative model. Firstly, we designed $g(\mathbf{N})$ to output logits for a $C \times 2 \times 2 \times S_{intra}$ tensor representing the logits of $\prod^{C,H=2,W=2} c, h, wp(\text{T}c, h, w)$. From this distribution, we sample $C \times 2 \times 2$ topological indices and then expand them over the spatial dimensions to form a $C \times H \times W$ tensor of $\hat{T}_{c,h,w}$. This approach effectively makes every node within the 2×2 tensor adjacent, thereby reducing the likelihood of optimization collapse.

Secondly, to circumvent the collapsing issue and locate a more favorable starting point for optimization, we initially train a context model with random characteristics akin to the approach in [13]. Instead of utilizing a random mask, we employ random topological indices for $\hat{T}c, h, w$. This preliminary training phase lasts about 100,000 iterations without including $\mathcal{L}\mathbf{G}_{intra}$ in the loss function. Subsequently, we initiate the Monte-Carlo optimization by incorporating $\mathcal{L}_{\mathbf{G}_{intra}}$, guiding the model towards improved local optima.

B.2 Training and Evaluation

Optimization Recall that we use the loss function Equation 19:

$$\mathcal{L} = \mathcal{L}_R + \lambda_D \mathcal{L}_D + \lambda_C \mathcal{L}_C + \mathcal{L}_{\mathbf{G}_{intra}} \tag{23}$$

Specifically, the rate loss, $\mathcal{L}_R$, integrates all latent variables within $\mathbf{L}$ as:

$$\mathcal{L}_R = \mathbb{E}_{\hat{\mathbf{y}} \sim q(\mathbf{y}|\mathbf{x}, \mathbf{G}_{inter}^{\mathbf{y},\mathbf{x}}), \hat{\mathbf{z}} \sim q(\mathbf{z}|\mathbf{y}, \mathbf{G}_{inter}^{\mathbf{z},\mathbf{y}})} \log_2 p(\hat{\mathbf{y}}|\hat{\mathbf{z}}, \mathbf{G}_{inter}^{\hat{\mathbf{y}},\hat{\mathbf{z}}}, \mathbf{G}_{intra}^{\hat{\mathbf{y}}}) p(\hat{\mathbf{z}}). \tag{24}$$

Distortion loss using MSE is given by:

$$\mathcal{L}_D = ||\hat{\mathbf{x}} - \mathbf{x}||_2, \quad \hat{\mathbf{x}} = \arg\max_{\hat{\mathbf{x}}} p(\hat{\mathbf{x}}|\hat{\mathbf{y}}, \mathbf{G}_{inter}^{\hat{\mathbf{x}},\hat{\mathbf{y}}}), \tag{25}$$

The complexity loss then encompasses the complexity of all BayesNets:

$$\mathcal{L}_C = \mathcal{L}_C(\mathbf{G}^{\mathbf{y},\mathbf{x}}_{inter}) + \mathcal{L}_C(\mathbf{G}^{\mathbf{z},\mathbf{y}}_{inter}) + \mathcal{L}_C(\mathbf{G}^{\hat{\mathbf{y}},\hat{\mathbf{z}}}_{inter}) + \mathcal{L}_C(\mathbf{G}^{\hat{\mathbf{x}},\hat{\mathbf{y}}}_{inter}) + \mathcal{L}_C(\mathbf{G}^{\hat{\mathbf{y}}}_{intra}). \quad (26)$$

where $\mathcal{L}_C(\mathbf{G}_{inter})$ can be computed as in Equation 11. Given that masked convolution is implemented as per Algorithm 1, $\mathcal{L}_C(\mathbf{G}^{\hat{\mathbf{y}}}_{intra})$ is constant and thus omitted from the loss function. Finally, with $\mathbf{G}_{intra}$ applied only to $\hat{\mathbf{y}}$ in this instance, the VIMCO loss for optimizing $\mathbf{G}_{intra}$ is:

$$\mathcal{L}_{\mathbf{G}_{intra}} = \mathbb{E}_{q(\mathbf{G}^{\hat{\mathbf{y}},1:K}_{intra})} \log \frac{1}{K} \sum_{i=1}^{K} \log p(\hat{\mathbf{y}}|\hat{\mathbf{z}}, \mathbf{G}^{\hat{\mathbf{y}},\hat{\mathbf{z}}}_{inter}, \mathbf{G}^{\hat{\mathbf{y}},i}_{intra}) \quad (27)$$

Training Schedule Typically, training for lossy image compression within the realm of NIC demands millions of iterations, which can be prohibitively time-intensive. To mitigate this training overhead, we utilize the pre-trained models of [25] available on CompressAI[7] as a starting point for our model's initialization. With this foundation, we proceed to substitute the original backbone with our inter-node BayesNet, which leverages slimmable models, and perform fine-tuning over approximately 500,000 iterations. Following this phase, we fix the backbone and shift our focus to training the newly integrated learned intra-node BayesNets—with configurations of 2-stage, 4-stage, and 10-stage—for an additional 500,000 iterations. Adopting this streamlined training regime, we emulate CompressAI by employing four different distortion weights, $\lambda_D = [0.0018, 0.0035, 0.0067, 0.0130]$, allowing us to craft a suite of models that cater to a spectrum of bit-rate preferences.

Complexity Metrics In our experimental evaluation, we concentrate on two principal metrics to assess computational complexity: (1) the total Multiply-Add Computation (MAC), a commonly accepted indicator of neural network computational load, and (2) the decompression speed, which is crucial in real-world applications where decompression is typically performed on edge devices with constrained processing power. We utilize ptflops[8] to tally the MAC figures. For decompression speed, we measure by calculating the ratio of the total size of the decompressed image data (in MegaBytes) to the aggregate time taken (in seconds) to decompress the entire image dataset. It's important to note that our image-loading routine is grounded in torchvision[9], which, by standard practice, loads images into 32-bit floating point tensors. Consequently, the volume of bytes in the decompressed images is quadrupled, mirroring the dimension count. As a result, the decompression speeds we report are effectively quadruple those cited in typical benchmarks.

C Complexity Level Greedy Search

This section presents a detailed breakdown of the BayesNet parameters employed by our proposed model across various complexity tiers, as well as the associated

[7] `https://interdigitalinc.github.io/CompressAI/zoo.html`
[8] `https://github.com/sovrasov/flops-counter.pytorch`
[9] `https://pytorch.org/vision/stable/index.html`

complexity metrics derived from a greedy search, as shown in Table 4. To align the complexity levels with performance metrics (specifically, the R-D loss), we first compute these for each potential combination of BayesNet parameters. We then establish 16 discrete complexity targets for the greedy search, spanning the range from the lowest to the highest computational complexity through linear interpolation. Under each target constraint, we select the BayesNet parameter set that yields the most favorable performance. For real-world deployment, our compilation of BayesNet parameters and their corresponding complexity metrics can guide the selection of models that meet specific computational constraints. In Table 4, we show an example model used in latter experiments in Section D.1, where the parameters and the resultant complexity statistics for each of the predetermined complexity levels are listed.

D Extra Experimental Results

D.1 Lossy Image Compression Performance Under Computation Constraints

In this section, we explore the primary domain of computationally scalable NIC—lossy image compression constrained by computational resources. Recognizing that most existing NIC research does not account for computation scalability, we have conducted a thorough analysis of the computational complexity of these works. We then selected specific models that align with predetermined computational constraints for our comparisons.

We evaluate our models under two distinct total MAC constraints: low MAC (up to 200 kMACs/pixel), which includes the baseline model Hyperprior [4], and high MAC (up to 500 kMACs/pixel), a range that encompasses many cutting-edge approaches. Within the 200 kMACs/pixel limit, we compare the low-bitrate version of Hyperprior (Balle2018-M192-N128, at 199 kMACs/pixel) [4], and SlimCAE with 96 channels (SlimCAE-96, at 114 kMACs/pixel) [30]. For the 500 kMACs/pixel threshold, several higher complexity models serve as benchmarks: the high-bitrate version of Hyperprior (Balle2018-M320-N192, at 442 kMACs/pixel) [4], the high-bitrate Joint Autoregressive model (Minnen2018-M320-N192, at 494 kMACs/pixel) [4], the low-bitrate version of Cheng2020 (Cheng2020-MN128, at 427 kMACs/pixel) [8], the low-bitrate Checkerboard Context model (He2021-MN128, at 436 kMACs/pixel) [13], and SlimCAE equipped with 192 channels (SlimCAE-192, at 433 kMACs/pixel) [30]. It is important to note that we reimplemented SlimCAE within our framework without λ-scheduling to maintain a consistent MAC across all bitrate levels. The comparative rate-distortion performance is illustrated in Figure 5.

The results indicate that, in the low MAC scenario, our method has comparable performance with Hyperprior and outperforms SlimCAE under the same MAC budget. SlimCAE's significantly lower MAC is due to its restricted adjustability, offering only five levels of channel width. In contrast, our proposed implementation can vary channel widths across five levels for each of the four

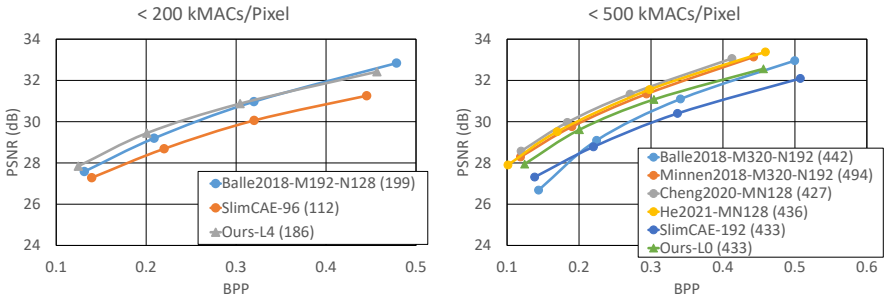


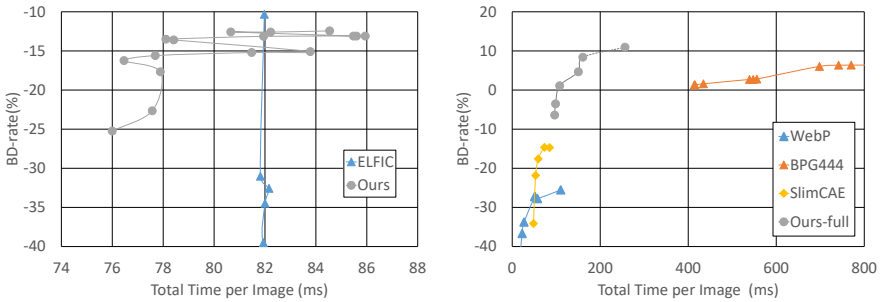
Fig. 5: Comparative results subject to total MACs constraints. The parenthetical value following each method in the legend denotes its MACs value in kMACs/Pixel.

neural networks, yielding up to $5^4 = 625$ potential computation levels, thereby enhancing its flexibility to meet MAC constraints. In the high MAC context, our implementation outperforms both SlimCAE and Hyperprior with a comparable MAC budget. However, it is outperformed by other state-of-the-art models, likely due to their superior neural network architectures and more extensive training regimes. This underscores the necessity to refine neural network designs and training strategies within our framework. Nonetheless, given the proposed framework's adaptability to a range of computational capabilities, a slight trade-off in performance might be considered reasonable.

D.2 Time comsumption comparison

In this section, we evaluate computational scalable codecs under the speed metric. Specifically, we first provide experimental results under total compression and decompression time per Kodak image (512x768 pixels, RGB format) and BD-Rate metric in Figure 6a under the same setup as Section 6.2. We include ELFIC [37] with the same backbone as comparison, which use universal slimmable network for synthesis transform network g_s. Moreover, we compare our full framework (same as Section D.1) with traditional scalable codecs WebP [29] and BPG(444) [6] as well as SlimCAE in Figure 6b.

Comparing with traditional codecs, our compression performance is similar to BPG with faster speed because of parallel computation on GPU, and having much better compression performance with relatively lower speed than WebP. Note that our scalability span is similar to traditional codecs and wider than other NIC codecs, mainly because of scalable autoregressive models with intra-node BayesNet which could better affect time comsumption. In comparison, ELFIC could only scale g_s which has little impact on the time consumption because of GPU acceleration, resulting in a limited scalability span.



(a) No AR. **(b)** Full model.

Fig. 6: Comparison for computational scalable frameworks subject to total time consumption.

D.3 Ablation Studies for Inter-node BayesNet

In this section, we include some further ablation studies for inter-node BayesNets. Specifically, we first compare the proposed heterogeneous parameterization for DAG in Equation 10 with the naive parameterization in Equation 8 in order to show that heterogeneous parameterization is more suitable for NIC. To this end, we implement another inter-node BayesNet with group convolution and group GDN for the naive parameterization. Specifically, we replace all convolutional layers as group convolution and GDN layers as group GDN. Note that as group convolution has different format of parameters with slimmable convolution, which cannot use existing pretrained models, we just train both backbones without autoregressive models from scratch by around 200k iterations, similar to Section 6.2. Note that as group convolution cannot be used as dynamic neural networks, we treat both backbones as static networks (for slimmable networks we just use the full channel width) and compare their performance accordingly. Secondly, we replace the adopted slimmable backbone as implementation for heterogeneous DAG with a static backbone, in order to show the impact on compression performance by this computation scalability implementation. Similarly, we also use the full channel width for slimmable networks and compare the compression performance with an alternative non-scalable implementation trained with static backbones. The results are shown in Figure 7.

Comparing the Heterogeneous DAG with the Naive one We could see that the naive parameterization performs much worse than the proposed heterogeneous parameterization, presumably because GDN cannot share information between channel groups, as discussed in Section 4.1.

Comparing Slimmable Backbone and Static Backbone The trained slimmable backbone performs slightly worse than the static counterpart. This is predictable as dynamic networks are generally harder to converge. This may could be improved in future works by applying better training tricks for slimmable networks, such as the sandwich rule and inplace distillation applied in universally slimmable network [37].

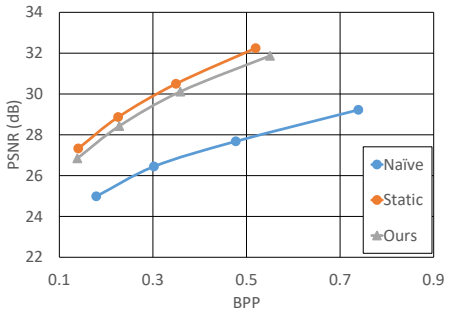


Fig. 7: Ablation study results for inter-node BayesNets.

D.4 Ablation Studies for Intra-node BayesNet

In this section, we delve into additional ablation studies concerning the intra-node BayesNets, focusing on key hyperparameters related to multipartite graph learning, such as the number of channel groups C and the kernel size used in dynamic masked convolutions. As in Section 6.3, we maintain a frozen backbone while training the learnable BayesNets and evaluate each variation based on BPP and decompression speed. Moreover, we provide visual representations of the partite topological indices. These results are documented in Table 5.

The data indicates that for a 2-stage structure, $C = 2$ yields better results than $C = 4$. In a 4-stage configuration, both $C = 2$ and $C = 4$ deliver comparable outcomes, while $C = 6$ underperforms slightly. The choice of C appears to have a modest effect on performance; however, aligning C with the number of parallel stages may be advantageous.Examining the visualized graphs, the arrangement for $C = 2$ resembles a checkerboard context model with a distinct channel-wise distribution of the black and white patterns—a formation we could term an "interlaced checkerboard." Meanwhile, the graph for $C = 4$ presents a hybrid of checkerboard and channel-wise autoregressive patterns, which is less effective than its interlaced counterpart.

When it comes to the kernel size, neither the 3×3 nor the 7×7 options surpass the performance of the widely implemented 5×5 kernels, a finding consistent with observations from the MaskConv context model [25]. Although without good performance, the learned topological indices also showcases an interlaced pattern, which indicate its effectiveness.

In summary, the optimal selection for the number of channel groups, C, generally corresponds to the number of processing stages, while the kernel size, K, should typically be set to 5. Most of the learned intra-node BayesNets exhibit an interlaced pattern, irrespective of the hyperparameters. This pattern is likely to capture more comprehensive context information by integrating cues from both the channel and spatial dimensions. Interestingly, the interlaced approach resonates with the concepts presented in ELIC [12], which amalgamates channel-wise autoregressive models with a checkerboard context model. This convergence

of strategies serves to underscore the efficacy of our proposed learning-based autoregressive model and offers valuable direction for the development of context models in the field of NIC.

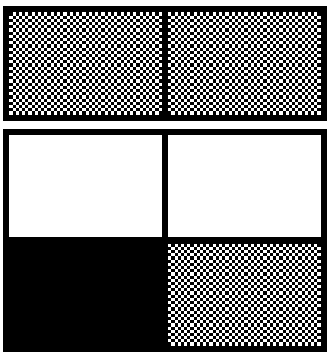
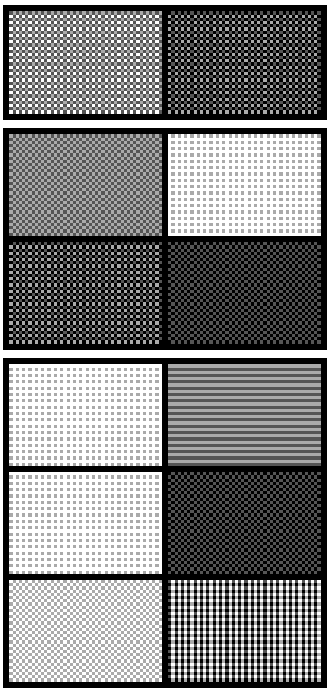
E Future Works

Despite the advances, the proposed method has limitations that should be addressed in future research. For instance, efficient techniques are still needed to maintain computational complexity within user-specified limits during coding, as current methods rely on a greedy search to meet this goal. Future work could also explore integration with more advanced backbone architectures to enhance compression performance, given that our study only investigates a single exemplary implementation.

Table 4: Example Greedy Search Results. Each complexity level search models within the corresponding constraint (in kMACs/Pixel). The statistics and the BayesNet parameters of searched models are also listed in each level.

Level			Statistics			Parameters			
Complexity	Constraint	Bit-rate	BPP	PSNR	kMACs/Pixel	$\hat{\mathbf{G}}_{inter}^{\mathbf{y},\mathbf{x}}$	$\hat{\mathbf{G}}_{inter}^{\mathbf{z},\mathbf{y}}$	$\hat{\mathbf{G}}_{inter}^{\hat{\mathbf{y}},\hat{\mathbf{z}}}$	$\hat{\mathbf{G}}_{inter}^{\hat{\mathbf{x}},\hat{\mathbf{y}}}$
0	433.1	0	0.124	27.95	433.1	0	0	0	0
		1	0.201	29.61	433.1	0	0	0	0
		2	0.304	31.08	433.1	0	0	0	0
		3	0.456	32.56	433.1	0	0	0	0
1	376.2	0	0.124	27.91	286.1	0	4	2	1
		1	0.200	29.56	293.1	0	2	0	1
		2	0.304	30.99	292.8	0	3	0	1
		3	0.459	32.55	357.9	2	1	3	0
2	319.3	0	0.124	27.91	286.1	0	4	2	1
		1	0.200	29.56	293.1	0	2	0	1
		2	0.304	30.99	292.8	0	3	0	1
		3	0.457	32.48	284.9	0	4	3	1
3	262.5	0	0.125	27.90	255.1	1	1	0	1
		1	0.200	29.52	254.5	1	2	0	1
		2	0.306	31.00	245.5	1	3	4	1
		3	0.456	32.49	248.7	1	1	2	1
4	205.6	0	0.125	27.84	182.9	0	4	4	2
		1	0.201	29.44	184.1	0	1	4	2
		2	0.304	30.88	183.4	0	2	4	2
		3	0.457	32.42	188.0	0	4	1	2
5	148.8	0	0.125	27.84	146.2	1	0	4	2
		1	0.200	29.42	145.5	1	3	3	2
		2	0.306	30.89	146.5	1	1	3	2
		3	0.457	32.41	144.8	1	2	4	2
6	91.9	0	0.124	27.64	81.9	2	1	3	3
		1	0.202	29.29	81.9	2	1	3	3
		2	0.306	30.73	86.0	2	1	1	3
		3	0.459	32.20	83.1	2	1	2	3
7	35.1	0	0.122	27.39	35.1	4	4	4	4
		1	0.199	28.95	35.1	4	4	4	4
		2	0.302	30.33	35.1	4	4	4	4
		3	0.453	31.85	35.1	4	4	4	4

Table 5: Ablation study results for intra-node BayesNets.

Stages	C	K	BPP	Speed	Vis
2	2	5	**0.3105**	170.30	
	4	5	0.312	142.02	
	2	3	0.3135	193.41	
	2	7	0.312	153.93	
4	2	5	**0.3075**	96.72	
	4	5	**0.3075**	80.96	
	6	5	0.3081	69.21	