APPENDIX A
MORE COMPLEXITY ANALYSIS

A.1 Time Complexity

In this section, we provide more details for the time complexity estimation described in Table 1. We will follow the phase partitioning (linear, preprocess, `MatMul`) described in Section 4 in the following analysis for clarity. The analysis of `MatMul` phase is performed on a single attention head, because all operations in this phase are parallelized on the head dimension. We omit the $\mathcal{O}(\cdot)$ notation for simplicity.

Vanilla. For a single head, Vanilla uses three weight matrices $\mathbf{W}^Q, \mathbf{W}^K, \mathbf{W}^V \in \mathbb{R}^{e\times d}$ to project hidden states $\mathbf{X} \in \mathbb{R}^{n\times e}$ into $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{n\times d}$, respectively. Here h heads are performed in parallel. Therefore, these projections contribute $3h \times ned = 3e^2n$ (note that $e = hd$ in common practice) in complexity for the whole linear phase. Note that, most of the efficient Transformers have three linear projections on $\mathbf{X}$ to produce $\mathbf{Q}, \mathbf{K}, \mathbf{V}$, thus having the same complexity in linear phase as Vanilla. We will only mention the difference in linear phase for rest of the models. Vanilla does not have preprocessing steps. The `MatMul` phase consists of two matrix multiplications $\mathbf{QK}^\top$ and $\mathbf{AV}$, where $\mathbf{A} = \texttt{softmax}(\mathbf{QK}^\top/\sqrt{d}) \in \mathbb{R}^{n\times n}$. Each multiplication has complexity dn^2, resulting in a total of $2dn^2$.

Longformer. In preprocessing phase, Longformer prepares the mask for sliding window attention. The mask inherits from the original padding mask, but is transformed for chunk-wise matrix multiplication in sliding window attention. In `MatMul` phase, each row in the sliding window attention has at most w positions to be calculated. g additional global columns are calculated for global pattern. The theoretical complexity of this phase is therefore $2(g + w)dn$. Here $(g + w)$ replaces n in the complexity of `MatMul` phase of Vanilla, which represents the sparsity of Longformer.

BigBird. BigBird performs operations at the granularity of basic blocks of size $b \times b$. In preprocessing phase, BigBird generates random positions for random attention at the granularity of basic blocks. It prepares masks for random and window attentions in this phase as well. In `MatMul` phase, BigBird calculates attention score for $(g + w + r)$ basic blocks in every b rows. The complexity is thus $2(g + w + r)bdn$.

SparseTrans. SparseTrans divides $\mathbf{Q}, \mathbf{K}$ into blocks of size b as well. It prepares masks for local and global attentions in block-transformed shapes in preprocessing phase. When calculating the attention score, b queries in the same block have b local keys and some global keys from other blocks, each of which provides g global keys. The number of other blocks is $(\frac{n}{b} - 1)$. Therefore, the complexity of this phase is $2(b + (\frac{n}{b} - 1)g)dn$.

Reformer. Reformer shares query and key weight matrices, which saves a linear projection compared with Vanilla, resulting in the complexity $2e^2n$ for linear phase. Reformer uses a random matrix $\mathbf{R} \in \mathbb{R}^{d\times s/2}$ to project one query (key) x to $x\mathbf{R}$ for hashing with the complexity $0.5sd$. This procedure is performed on $h\times n$ queries (keys) with l rounds in parallel, resulting in the complexity $0.5sd\times hnl = 0.5lsen$ for preprocessing phase. In `MatMul` phase, Reformer splits queries (keys) into uniform chunks of size $c = 2n/s$. Each query chunk attends to two key chunks (one is the local key chunk and the other one is the key chunk before the local one). The complexity of chunk-wise $\mathbf{QK}^\top$ is thus $2cdn = 4dn^2/s$. The chunk-wise $\mathbf{AV}$ has the same complexity. This is for one hash round. l rounds of output vectors will be weighted averaged according to logits, and the overall complexity of `MatMul` phase is $2l \times 4dn^2/s = 8ldn^2/s$. By specifying chunk size $c = 2n/s$ with a constant value, `MatMul` phase has linear complexity to n. In practice, the number of buckets s is limited under $2\max(c, \sqrt{n/c})$. Otherwise, Reformer defines two sets of buckets with s_1, s_2 buckets in each set respectively ($s_1s_2 = s$), and hashes vectors into these buckets. It uses indices of hashing buckets to form a tuple $(i_1, i_2) \in \{1, 2, \ldots, s_1\} \times \{1, 2, \ldots, s_2\}$, which corresponds to $s_1s_2 = s$ buckets. This technique further reduces the complexity of hashing when s is large.

ClusterAttn. In preprocessing phase, ClusterAttn groups queries into c clusters. It first hashes queries into scalars and then performs K-means clustering to these scalars with Hamming distance. Hash projections and l iterations of K-means have an overall complexity $en + lcn$. In `MatMul` phase, c query centroids attend to all keys, which costs cdn. The improved version of ClusterAttn further selects top-k keys to attend to queries in the same cluster, which costs kdn. These two parts have an overall complexity $(c + k)dn$ for the $\mathbf{QK}$ part. The complexity is doubled after taking the $\mathbf{AV}$ part into account.

LinearAttn. As a kernelization method, LinearAttn applies the feature map $\phi(\cdot) = \texttt{elu}(\cdot) + 1$ on $\mathbf{Q}, \mathbf{K}$. It is an element-wise activation that is not comparable to matrix multiplication in complexity. By switching the calculation order from $(\mathbf{QK}^\top)\mathbf{V}$ to $\mathbf{Q}(\mathbf{K}^\top\mathbf{V})$, we have $\mathbf{K}^\top\mathbf{V} = \mathbf{A}' \in \mathbb{R}^{d\times d}$, avoiding to generate a full attention matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$. The overall complexity of `MatMul` phase is $2d^2n$.

Cosformer. Cosformer not only applies feature map on $\mathbf{Q}, \mathbf{K}$, but uses trigonometric function to re-weight these values as well. These are all included in preprocessing phase. The re-weighting mechanism requires Cosformer to calculate the two formulas in E.q. (10). Therefore, the complexity of Cosformer in `MatMul` phase is twice that of LinearAttn.

Performer. In preprocessing phase, before applying feature map, Performer uses a set of Gaussian basis $\Omega \in \mathbb{R}^{p\times d}$ to project $\mathbf{Q}, \mathbf{K}$ into the shape of (n, p). This contributes $2h \times npd = 2pen$ to complexity for all heads. In `MatMul` phase, queries and keys after projection are of shape (n, p), while values keep the shape (n, d). As a result, the kernelized attention calculation has complexity $2pdn$.

Linformer. The calculation for $\mathbf{P}_1\mathbf{K}$ and $\mathbf{P}_2\mathbf{V}$ in E.q. (12) has a complexity of $2h\times pdn = 2pen$ in total. This is the complexity of preprocessing phase. The attention calculation for $\mathbf{Q}, \mathbf{P}_1\mathbf{K}, \mathbf{P}_2\mathbf{V}$ has complexity $2pdn$ for each head.

Nyströmformer. Nyströmformer consists of six matrix multiplications for attention calculation in E.q. (13). It first calculates $\mathbf{Q}\tilde{\mathbf{K}}^\top, \tilde{\mathbf{Q}}\tilde{\mathbf{K}}^\top$ and $\tilde{\mathbf{Q}}\mathbf{K}^\top$. This costs $pdn + p^2d + pdn$. It then calculates $\mathbf{A}_k\mathbf{A}_{qk}^+$ and $\mathbf{A}_q\mathbf{V}$, which costs $p^2n + pdn$. The last multiplication between $\mathbf{A}_k\mathbf{A}_{qk}^+$ and $\mathbf{A}_q\mathbf{V}$ costs pdn. Therefore, the overall complexity of `MatMul` phase is $4pdn + p^2d + p^2n$. One round of Moore-Penrose pseudoinverse approximation has 4 matrix multiplications between two $p \times p$ matrices on a single head. Therefore, l

rounds of approximation costs $hl \times 4p^3 = 4hlp^3$. This is the complexity of preprocessing phase.

Synthesizer. The factorized random variant of Synthesizer takes no inputs for its attention matrix, resulting in only one projection for $\mathbf{V}$ in lienar phase. The complexity of linear phase is thus reduced to e^2n. Synthesizer has no preprocessing steps. And in `MatMul` phase, $\mathbf{R}_1\mathbf{R}_2$ in E.q. (14) costs pn^2. Synthesizer generates a full attention matrix of shape $(n \times n)$, resulting in complexity dn^2 for $\mathbf{AV}$ part. The overall complexity of `MatMul` phase is $(p + d)n^2$.

TransNormer. TransNormer has two parts, DiagAttention and NormAttention. DiagAttention is a sparsification method. It needs to prepare block masks in preprocessing phase, and block-local attention with block size b costs $2bdn$ in `MatMul` phase. NormAttention can be viewed as a kernelization method. It applies feature map on $\mathbf{Q}, \mathbf{K}$ in preprocessing phase and follows the calculation scheme of kernelization methods, which costs $2d^2n$ in `MatMul` phase.

APPENDIX B
EXPERIMENTAL SETUP DETAILS

LRA. We use the conventional Transformer with softmax attention as the baseline, which is also termed as Vanilla in this work. Every efficient method is implemented on the same Transformer backbone, except that we replace the self-attention module with the corresponding efficient attention module. All methods, including the baseline, use a Transformer model with 2 layers and 2 attention heads. Transformer hidden dimension and FFN hidden dimension are 64, 128 respectively for Image and Pathfinder tasks, and are 128, 256 respectively for the rest of the tasks.

NLP. We adopt masked language modeling (MLM) as the pre-training task, and take RoBERTa as the baseline and the backbone for all other methods. We follow the pre-training and fine-tuning protocol of RoBERTa. For the sake of saving time and computation consumption (it is well known that pre-training on large models is expensive in time and computation), we use the tiny variant of RoBERTa as the backbone model, with 4 layers, 256 embedding and Transformer hidden dimension, 1024 FFN hidden dimension, and 4 attention heads. CLS token is deployed in all tasks, CLS pooling is used by default in downstream tasks. We pre-train the models for 190 epochs with batch size 256 (approximately 150k steps), and fine-tune them for 5 epochs with batch size 32 on each downstream task. Input sequence length is 512.

CV. We adopt SimMIM as the framework of masked image modeling for image pre-training. ViT is the baseline and the backbone of all compared efficient Transformers. A larger model scale is used for ViT, with 8 layers, 512 embedding and Transformer hidden dimension, 2048 FFN hidden dimension, and 8 attention heads. We set additional configurations: image resolution 64×64, patch size for ViT is 4, mask patch size for SimMIM is 8, and CLS token is used. We pre-train the models for 150 epochs with batch size 512, and fine-tune them for 100 epochs with batch size 512 on the downstream task. For fine-tuning, we run a grid search over learning rates from $\{0.001, 0.0005, 0.0002\}$ and report the best accuracy. Input sequence length is 256.

Different sequence length. We perform the experiment in NLP MLM pre-training task as many prior works use it to test the stability of Transformer as sequence length changes. In this task, the model is trained for one whole epoch if all tokens of the training corpus are fed into the model. We increase sequence length from 512 to 2048, while decrease the batch size from 256 to 64. This makes the number of tokens for one training step unchanged, which results in identical number of rounds of gradient updates in one epoch. 250-epochs training (approximately 200k steps) is employed for sequence length of 2048. We also keep the compute budget unchanged for different sequence lengths, which means the hyper-parameters associated with the methods are kept the same. SparseTrans is the only exception because its compute cost is actually a quadratic function of sequence length.

APPENDIX C
MORE EXPERIMENT RESULTS

C.1 Pre-training Tasks

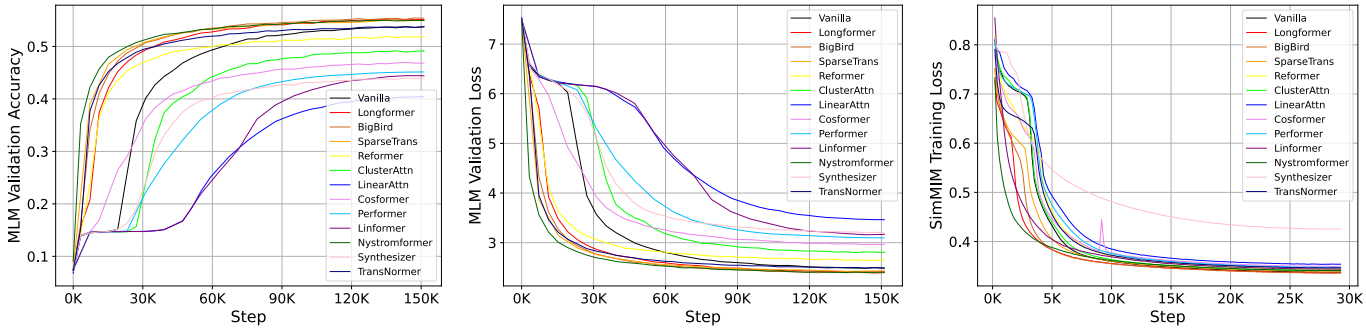
For MLM pre-training task, we illustrate the curves of accuracy and loss on the validation set for all efficient Transformers along with the vanilla Transformer, as shown in Figures 7a and 7b respectively. For SimMIM pre-training task, we use the whole Tiny-ImageNet-200 dataset for training, thus illustrating training loss curves in Figure 7c.

C.2 Convergence Results on Different Sequence Lengths

We report MLM validation loss throughout the training progress on different sequence lengths for comparison. The results of all efficient Transformers as well as the vanilla Transformer are shown in Figure 8.

APPENDIX D
MORE INTERPRETABILITY STUDY

We provide the attention visualization results of different Transformer variants for all 4 heads and 4 layers in Figures 9-21. The attention values are scaled logarithmically for better illustration. As we can see, the attention score of Vanilla has strong local biases. Many large attention scores appear even far away from the diagonal area, which complements long-range dependencies to the diagonal local attention. Position-based sparsification methods apply sparse patterns on attention matrices, and blank positions in the visualization are not included in attention calculation. Their local sparse patterns formulate a diagonal region in the figures. There are also some dark rows and columns to learn the long-range dependencies in these sparse attentions. Longformer only uses sliding window attention in the MLM task, and BigBird puts global blocks at the edges of attention matrices, while SparseTrans has a global column after each block. For content-based sparsification methods, their illustrated local and long-range dependencies change significantly. show the effectiveness of their hashing and/or clustering estimation. Both Reformer and ClusterAttn learn the dependencies in the low-dimensional space after hashing and/or clustering, rather than the original space. That's why there are no significant dark diagonal



(a) MLM validatation accuracy (b) MLM validatation loss (c) SimMIM training loss

Fig. 7: Results on pre-training tasks

blocks in the attention matrices. Kernelization methods reorder the multiplication operators and do not have explicit attention anymore. But we can still obtain the approximate full attention by recomputing $\phi(\mathbf{Q})\phi(\mathbf{K})^\top$ first and divide it by the denominator of E.q. (7). We also observe some local biases, especially for Cosformer and Performer due to their additional optimizations (e.g., re-weighting). For factorization methods, we use the computation results before $\mathbf{V}$ in E.q. (12) and (13) as their attention matrices. The results involve some negative values due to $\mathbf{P}_2$ and $\mathbf{A}^+_{qk}$, which are represented as the blank positions in Figures 5j and 5k. The random attention matrix of Synthesizer, which is independent of inputs and thus identical for all data instances, has broader local regions for high attention values. As a combination method, TransNormer uses DiagAttention to model local dependencies, while its NormAttention devotes to long-range dependencies and does not show a strong local trend as other kernelization methods do.

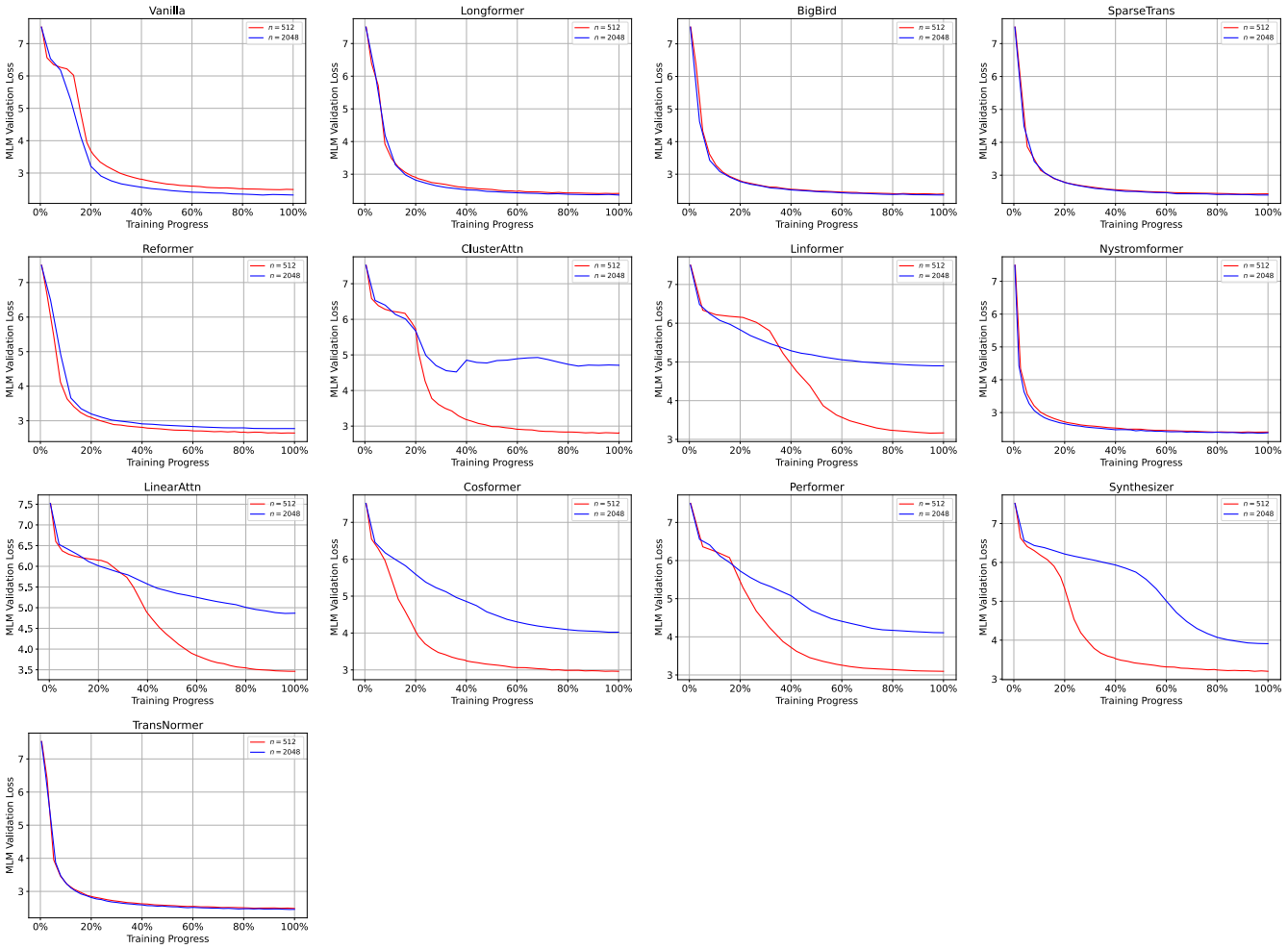
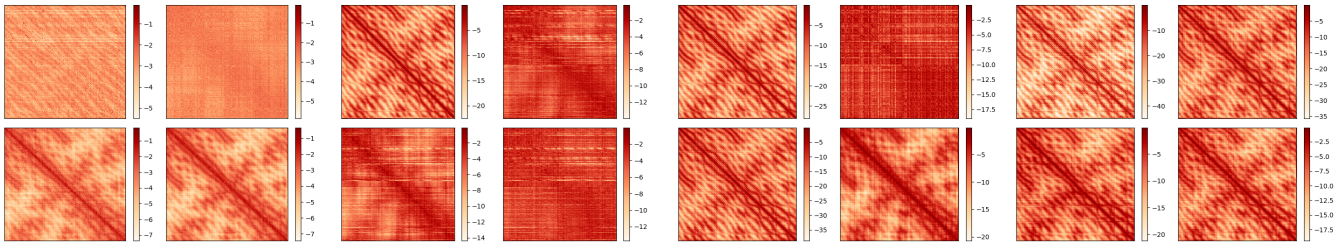
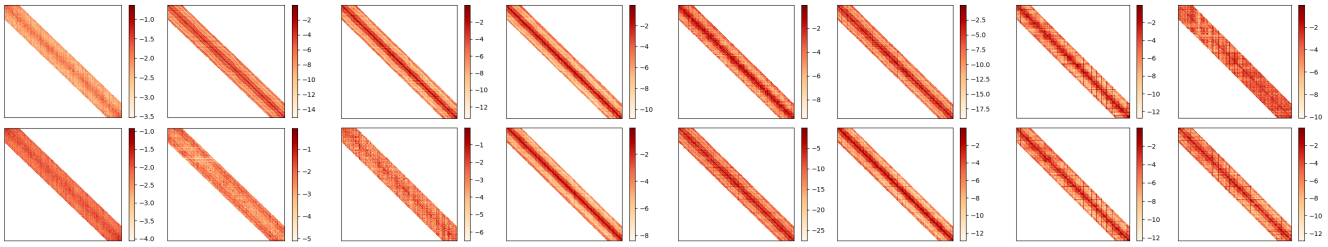


Fig. 8: MLM validation loss on different sequence lengths for efficient Transformers



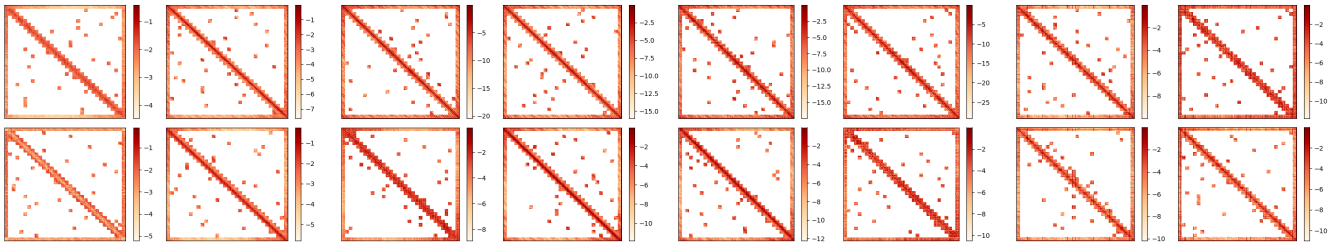
(a) Layer 1 (b) Layer 2 (c) Layer 3 (d) Layer 4

Fig. 9: Visualization of attention matrices for all attention heads in Vanilla Transformer



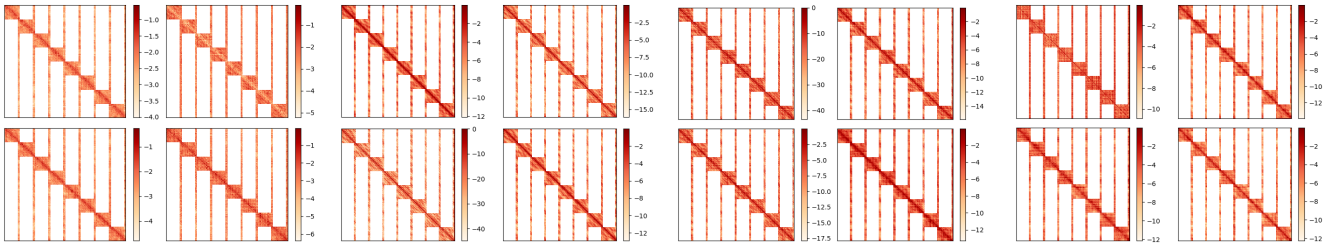
(a) Layer 1 (b) Layer 2 (c) Layer 3 (d) Layer 4

Fig. 10: Visualization of attention matrices for all attention heads in Longformer



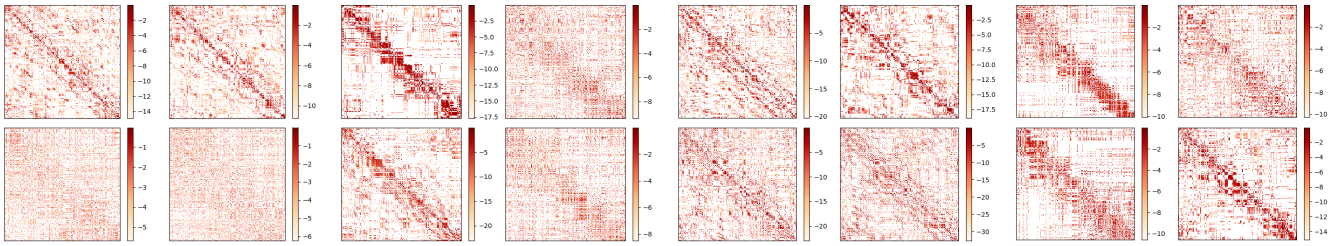
(a) Layer 1 (b) Layer 2 (c) Layer 3 (d) Layer 4

Fig. 11: Visualization of attention matrices for all attention heads in BigBird



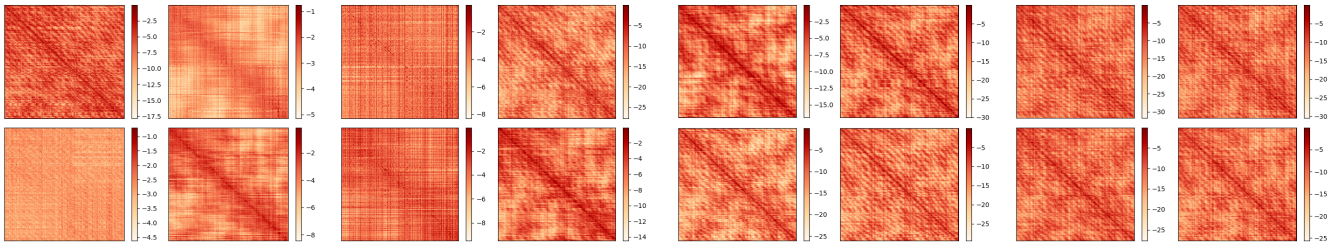
(a) Layer 1 (b) Layer 2 (c) Layer 3 (d) Layer 4

Fig. 12: Visualization of attention matrices for all attention heads in SparseTrans



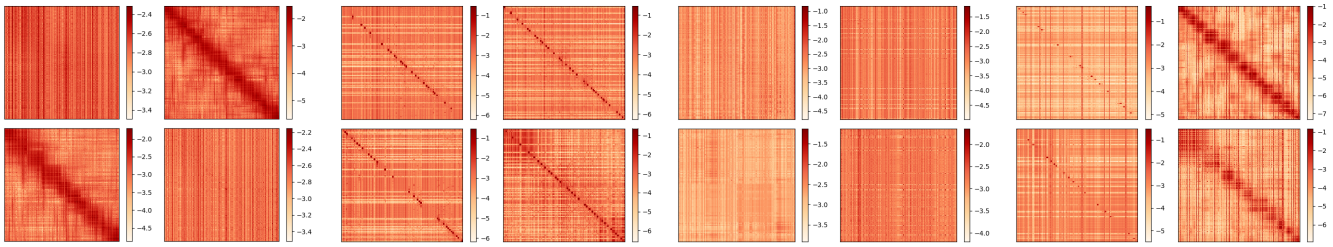
(a) Layer 1 (b) Layer 2 (c) Layer 3 (d) Layer 4

Fig. 13: Visualization of attention matrices for all attention heads in Reformer



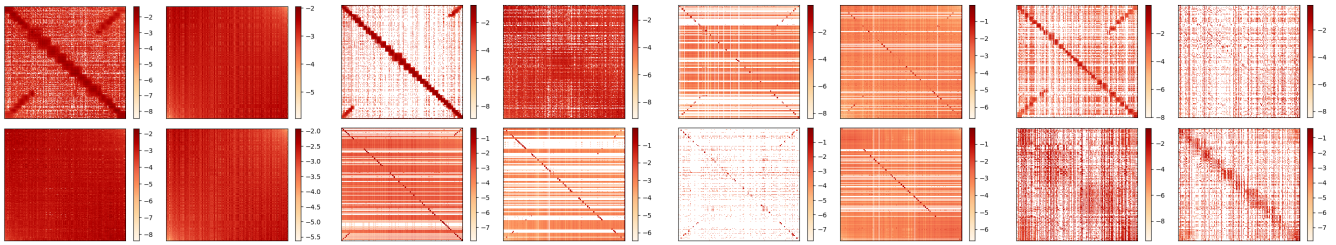
(a) Layer 1 (b) Layer 2 (c) Layer 3 (d) Layer 4

Fig. 14: Visualization of attention matrices for all attention heads in ClusterAttn



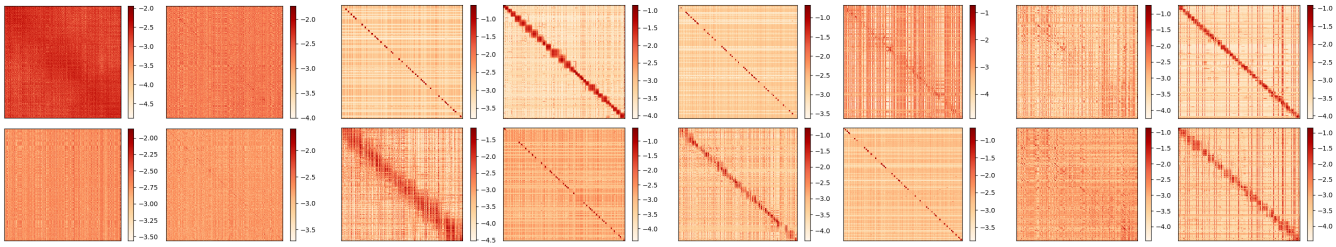
(a) Layer 1 (b) Layer 2 (c) Layer 3 (d) Layer 4

Fig. 15: Visualization of attention matrices for all attention heads in LinearAttn



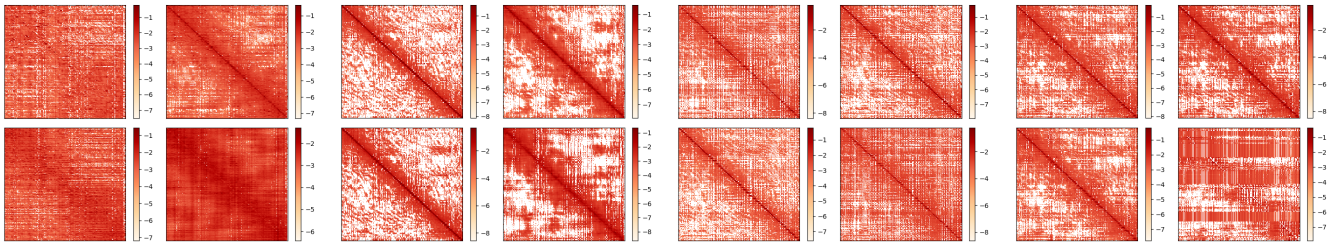
(a) Layer 1 (b) Layer 2 (c) Layer 3 (d) Layer 4

Fig. 16: Visualization of attention matrices for all attention heads in Cosformer



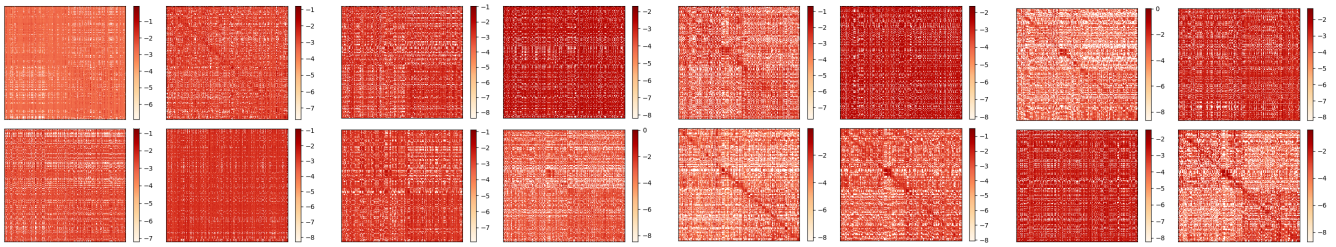
(a) Layer 1 (b) Layer 2 (c) Layer 3 (d) Layer 4

Fig. 17: Visualization of attention matrices for all attention heads in Performer



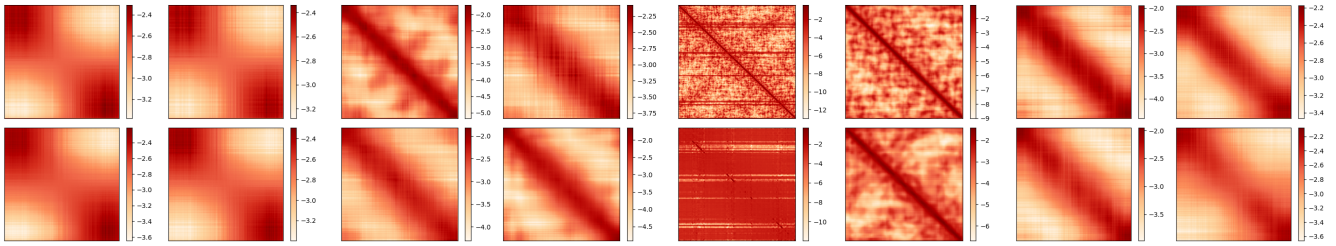
(a) Layer 1 (b) Layer 2 (c) Layer 3 (d) Layer 4

Fig. 18: Visualization of attention matrices for all attention heads in Linformer



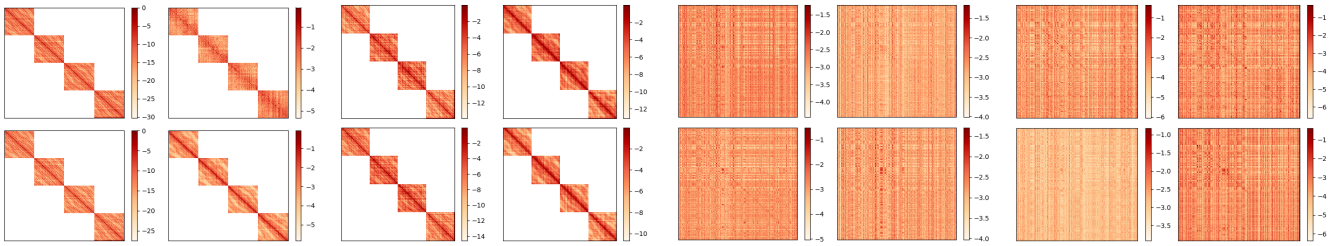
(a) Layer 1 (b) Layer 2 (c) Layer 3 (d) Layer 4

Fig. 19: Visualization of attention matrices for all attention heads in Nyströmformer



(a) Layer 1 (b) Layer 2 (c) Layer 3 (d) Layer 4

Fig. 20: Visualization of attention matrices for all attention heads in Synthesizer



(a) Layer 1 (b) Layer 2 (c) Layer 3 (d) Layer 4

Fig. 21: Visualization of attention matrices for all attention heads in TransNormer