Reviving Efficient Attention for Long Context Language Modeling: A Survey

[Technical Report]

1 PRELIMINARY

1.1 Overview of Transformer

The first Transformer model was proposed by Vaswani et al. [38], as well as the attention mechanism. Most Transformer models contain several identical Transformer modules (i.e., blocks) and each of them is usually a stack of several layers, including *multi-head self-attention* (MSA) layer, the *feed-forward network* (FFN) layer and other auxiliary structures (e.g., residual connection, layer normalization). Figure 1 illustrates the architecture of a single Transformer module and the visualization of the attention mechanism. Typically the input tensor of a Transformer consists of several data samples.[1] For each data sample (i.e., a sequence of tokens), we denote its tensor representation as $X \in \mathbb{R}^{n \times e}$, where n is the sequence length in terms of the number of tokens and e is the embedding size (i.e., hidden size). Each token is represented as an embedding vector that implies both its own semantic information and the context information inside the given sequence.

1.1.1 Multi-Head Self-Attention. The Transformer model employs a scaled dot-product attention mechanism in a multi-headed fashion. For each head, a standard scaled dot-product attention takes three inputs, ($\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{n \times d}$), representing queries, keys and values respectively. Its output is calculated as following:

$$\texttt{Attn}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \texttt{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d}}\right)\mathbf{V} = \mathbf{A}\mathbf{V}, \tag{1}$$

where softmax is applied in a row-wise manner. The attention mechanism first calculates the attention score matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ from the dot product of row vectors in $\mathbf{Q}$ and $\mathbf{K}$. Specifically, each position (i, j) in $\mathbf{A}$ implies the the strength of the relationship between the i-th token and the j-th token in the given sequence. After a row-wise normalization (by softmax) step, it is used to calculate the weighted summation over the input value vectors $\mathbf{V}$, which finally becomes the output of attention module.

In Transformer, a multi-head self-attention takes hidden vector matrix $\mathbf{X} \in \mathbb{R}^{n \times e}$ as the input. The calculation is then split into h different heads ($h = e/d$). The i-th head uses weight matrices $\mathbf{W}_i^Q, \mathbf{W}_i^K, \mathbf{W}_i^V \in \mathbb{R}^{e \times d}$ to project $\mathbf{X}$ into queries, keys and values, for scaled dot-product attention calculation. The outputs of all heads are then concatenated and projected back to e-dim hidden vectors.

$$\texttt{MultiHeadAttn}(\mathbf{X}) = \texttt{Concat}(\mathbf{H}_1, \ldots, \mathbf{H}_h)\mathbf{W}^O, \tag{2}$$

$$\text{where } \mathbf{H}_i = \texttt{Attn}(\mathbf{X}\mathbf{W}_i^Q, \mathbf{X}\mathbf{W}_i^K, \mathbf{X}\mathbf{W}_i^V) \tag{3}$$

1.1.2 Feed-Forward Network. The output of attention module will then be passed into a fully connected feed-forward network, where every position are operated independently. This network has two linear projection layers with a ReLU activation.

$$\texttt{FFN}(\mathbf{X}) = \texttt{ReLU}(\mathbf{X}\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2. \tag{4}$$

where $\mathbf{W}_1 \in \mathbb{R}^{e \times f}$ and $\mathbf{W}_2 \in \mathbb{R}^{f \times e}$ are two weight matrices.

1.1.3 Transformer Module. In the vanilla Transformer module (i.e., a encoder block [38]), there are two essential sub-layers including MSA and FFN. A residual connector is employed around each sub-layer to make the model deeper, followed by a layer normalization. As a result, with input $\mathbf{X}$, the module structure can be expressed as

$$\texttt{LayerNorm}(\mathbf{X} + \texttt{SubLayer}(\mathbf{X})), \tag{5}$$

where SubLayer can be the function of either MSA or FFN.

1.2 Efficiency Metrics

In this approach, we study the efficiency of recent efficient attention mechanisms in three aspects, including computation efficiency, memory efficiency, and statistical efficiency. Before introducing details of these efficient attention methods, we start by leveraging three efficiency measurements as well as their corresponding evaluation metrics in the following.

Computational Efficiency. The *computational efficiency* directly describes the practical execution speed of the Transformer models over existing DL systems. Basically, it could be measured by the total GPU kernel execution time when using GPU as the accelerator. The computation efficiency is mainly determined by the theoretical computation complexity of the Transformer model architecture. Besides, it is also affected by many hardware-level factors, e.g., threads parallelism [10], memory accessing [15], kernel launching and scheduling [48].

Memory Efficiency. We consider the *memory efficiency* by estimating the memory usage of the Transformer's execution. For common DL training systems, the memory usage includes model parameters, model gradients, and computation activations (e.g., inputs and outputs of each operator in the model) [28, 33]. The memory complexity could be used to approximate the pratical memory footprint. It can be optimized by adjusting the computation procedure to reduce the number of model parameters or activations.

Statistical Efficiency. The *statistical efficiency* [46] is first proposed to quantify the convergence performance for ML model training. Transformer models are often pre-trained over large amounts of input datasets and could be shared and reused on diverse downstream tasks by fine-tuning over small domain-specific datasets. In this approach, we further extend the definition of statistical efficiency to describe the model effectiveness on different downstream tasks through their evaluation metrics (e.g., test accuracy).

[1] The following analysis are from a single data sample's perspective to avoid inessential notations and it is easy to scale to a batch of data samples by adding a batch dimension.

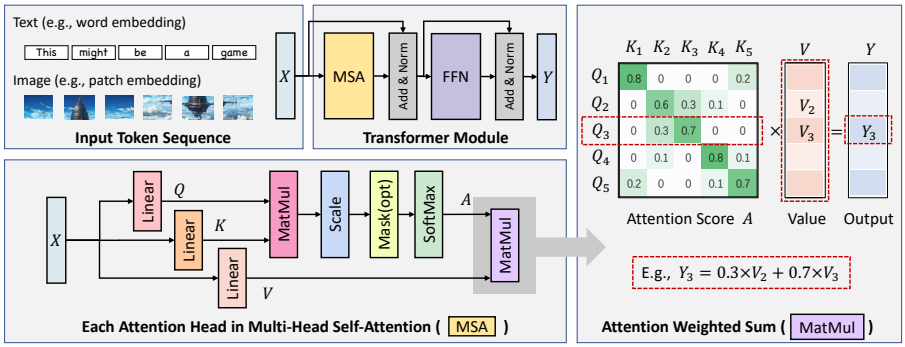


Figure 1: Illustration of Transformer module and Multi-Head Self-Attention

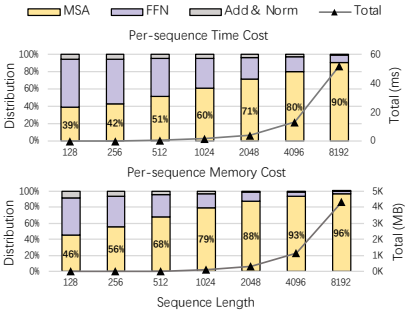


Figure 2: The increasing time and memory costs when scaling a Transformer module to longer sequences

1.3 Selected Efficient Attention Mechanisms

We now list all 12 selected efficient attention methods to be studied, and briefly introduces our taxonomy. We select Longformer [9], BigBird [45], SparseTrans [11] as representatives for positional selection methods, Reformer [19], ClusterAttn [39] for contextual compression methods, LinearAttn [17], Cosformer [47], Performer [12] for activation kernelization methods, Linformer [41], Nyströmformer [44] for low-rank factorization methods. In addition, we select Synthesizer [36] and TransNormer [30] as representatives for the category of **other approaches**. It should be noted that TransNormer is a combination of DiagAttention (a positional selection method) and NormAttention (an activation kernelization method).

2 COMPLEXITY ANALYSIS

In this section, we provide an in-depth complexity analysis of the above efficient attention methods. Since the attention calculation of these approaches varies a lot from each other, we generally divide the attention computation procedure into three phases. Note that, they are not necessarily to be executed in sequential order but are responsible for different functionalities in these approaches.

P1: Linear. This phase includes all linear projections on $\mathbf{X}$ to generate $\mathbf{Q}, \mathbf{K}$, and $\mathbf{V}$. In this phase, all models except Reformer and Synthesizer retain three projections with the time complexity of $O(3e^2n)$. Specially, Reformer makes keys equal to queries, which saves a weight matrix for keys, resulting in a 1/3 reduction in time complexity. Synthesizer's factorized random attention matrix takes no inputs, which saves two linear projections for $\mathbf{Q}, \mathbf{K}$.

P2: Preprocess. In this phase, efficient attention methods do their additional processes on $\mathbf{Q}, \mathbf{K}, \mathbf{V}$. For contextual compression methods, they use this phase to determine which query-key pairs are to be calculated (e.g., hashing and clustering). Activation kernelization methods and TransNormer-Norm apply feature maps ϕ on $\mathbf{Q}, \mathbf{K}$, while Cosformer re-weights them after mapping. Performer projects hidden vectors using Gaussian basis beforehand, and Linformer does low-dim projection on $\mathbf{K}, \mathbf{V}$. Nyströmformer calculates the Moore-Penrose pseudoinverse through the iterative algorithm.

P3: `MatMul`. This phase is the major source of computation costs in attention calculation, i.e. the matrix multiplications of $\mathbf{QK}^\top$ and $\mathbf{AV}$ in E.q. (1). Most of the efficient attention methods basically follow the multiplication semantics of three matrices except Nyströmformer, which consists of six matrix multiplications in its approximation equation full attention.

Complexity. Table 1 summarizes each phase's time complexity for these efficient attention methods. Note that, these complexity results are deduced based on the algorithm description and may be affected by the system implementation in practice (e.g., ignoring some inconsequential operations). More details about the complexity deduction are described in the Appendix. To provide a more intuitive comparison, we further show the numbers of three major operators including `MatMul` (MM Op), memory re-formatting (MR Op), and element-wise activation (EA Op). MM Op is the compute-intensive one while the other two are memory-intensive. Note that the memory complexity is not included since memory footprints are heavily influenced by the implementation decisions (e.g., in-place operations, recomputation). We will examine the memory consumption empirically in Section 3.

3 EXPERIMENTS

In this section, we conduct extensive experiments over five categories of efficient attention methods (including 14 representative state-of-the-art approaches in total) and manage to address the following research questions:

- **Q1.** How do these approaches affect the statistical efficiency (i.e., model performance) of Transformers on DL tasks in different fields?
- **Q2.** How does the input data sequence length impact the statistical efficiency of these approaches?
- **Q3.** How much computational and memory efficiency can be really improved when applying these techniques?
- **Q4.** What are the key findings behind our evaluation? How to make the right choice among these techniques in real data science problems?

3.1 Experimental Setup

Before starting our evaluations, we need to clarify the necessary experimental setup first (e.g., hardware, task settings, and so on).

Hardware. All experiments are conducted on the same hardware. The machine is equipped with two Intel Xeon Gold 5120 CPUs, 376

Table 1: Summary of time complexity ($O(\cdot)$) of efficient attention methods

Taxonomy	Model	Linear	Preprocess	`MatMul` (per attention head)	`softmax`	# MM Ops	# MR Ops	# EA Ops
	Full	$3e^2n$	–	$2dn^2$	✔	5	0	1
Positional Selection	Longformer	$3e^2n$	Prepare window masks	$2(g+w)dn$	✔	8	1	2
	BigBird	$3e^2n$	Prepare random masks	$2(g+w+r)bdn$	✔	19	15	5
	SparseTrans	$3e^2n$	Prepare block masks	$2(b+(\frac{n}{b}-1)g)dn$	✔	8	3	1
Contextual Compression	Reformer	$2e^2n$	$0.5lsen$	$8ldn^2/s$	✔	5	11	4
	ClusterAttn	$3e^2n$	$lcn+en$	$2(c+k)dn$	✔	6	>2	1
Activation Kernelization	LinearAttn	$3e^2n$	ϕ	$2d^2n$	✗	5	0	2
	Cosformer	$3e^2n$	ϕ, Re-weighting	$4d^2n$	✗	5	2	6
	Performer	$3e^2n$	$2pen$, ϕ	$2pdn$	✗	7	0	1
Low-Rank Factorization	Linformer	$3e^2n$	$2pen$	$2pdn$	✔	7	0	1
	Nyströmformer	$3e^2n$	$4hlp^3$	$4pdn+p^2d+p^2n$	✔	$9+4l$	0	3
Other Approaches	Synthesizer	e^2n	–	$(p+d)n^2$	✔	3	0	1
	TransNormer-Diag	$3e^2n$	Prepare block masks	$2bdn$	✔	6	0	1
	TransNormer-Norm	$3e^2n$	ϕ	$2d^2n$	✗	5	0	2

MM refers to `MatMul`, MR refers to Memory Re-formatting (e.g., `gather`,`scatter`,`concat and index_select`), and EA refers to Element-wise Activation (e.g., `ReLU`).

Table 2: Test Accuracy on LRA Benchmark, CLS Pooling

Model	ListOps	Text	Retrieval	Image	Pathfinder	Avg.
Full	37.18±0.12	66.04±0.16	81.15±0.19	39.61±1.41	73.29±0.96	59.46 4
Longformer	37.32±0.24	66.10±0.09	81.64±0.28	43.02±0.44	73.81±0.46	60.38 1
BigBird	38.27±0.31	65.55±0.04	80.28±0.47	42.66±0.21	73.80±0.20	60.11 2
SparseTrans	37.49±0.48	64.25±0.38	80.25±0.22	42.58±0.61	71.31±1.31	59.18 5
Reformer	36.11±0.70	61.30±1.05	78.54±0.05	36.82±0.26	70.31±0.35	56.62 9
ClusterAttn	37.32±0.06	66.11±0.09	80.56±0.11	38.21±1.08	54.75±1.08	55.39 10
LinearAttn	19.77±0.34	66.04±0.03	79.71±0.08	38.05±0.61	71.09±0.53	54.93 11
Cosformer	37.38±0.16	66.24±0.09	81.16±0.30	36.35±1.65	73.59±0.52	58.94 6
Performer	38.07±0.11	66.12±0.12	81.14±0.28	39.32±0.60	73.07±0.71	59.54 3
Linformer	37.63±0.21	55.37±1.85	78.23±0.05	39.34±0.31	62.12±6.77	54.54 12
Nyströmformer	37.45±1.26	63.70±0.40	79.36±0.34	39.81±0.37	71.01±0.49	58.27 7
Synthesizer	19.89±0.31	65.60±0.20	77.94±0.28	35.57±0.80	67.57±0.24	53.32 13
TransNormer	36.97±0.05	64.90±0.15	82.31±0.04	38.59±1.67	68.19±0.53	58.19 8

Table 3: Test Accuracy on LRA Benchmark, MEAN Pooling

Model	ListOps	Text	Retrieval	Image	Pathfinder	Avg.
Full	37.17±0.38	65.85±0.15	81.12±0.17	37.81±1.35	73.41±0.40	59.07 3
Longformer	37.01±0.28	65.72±0.14	81.00±0.12	41.26±0.35	70.32±0.38	59.06 4
BigBird	38.04±0.44	65.55±0.03	80.54±0.27	40.54±0.63	69.55±0.46	58.84 6
SparseTrans	37.55±0.21	64.04±0.51	79.82±0.27	43.49±0.87	71.65±0.23	59.31 2
Reformer	23.51±0.17	65.33±0.20	79.34±0.29	40.14±0.20	69.20±0.60	55.50 11
ClusterAttn	36.96±0.09	66.08±0.12	80.75±0.34	39.19±1.25	67.19±0.33	58.03 8
LinearAttn	20.57±0.57	66.06±0.02	80.18±0.30	38.83±0.75	72.61±0.54	55.65 10
Cosformer	37.48±0.12	66.14±0.08	81.88±0.26	35.74±0.59	73.14±0.44	58.87 5
Performer	37.52±0.22	66.18±0.07	81.60±0.07	39.40±0.32	72.75±1.32	59.49 1
Linformer	37.90±0.70	54.10±1.64	78.09±0.11	40.08±1.66	66.97±0.21	55.43 12
Nyströmformer	31.08±8.05	65.36±0.06	79.45±0.48	40.15±0.65	69.86±1.34	57.18 9
Synthesizer	18.87±0.91	65.56±0.20	78.96±0.16	35.46±1.27	67.58±0.18	53.28 13
TransNormer	37.35±0.41	64.76±0.13	82.49±0.07	38.47±0.97	68.40±0.48	58.29 7

GB DRAM and 8 NVIDIA Titan RTX 24GB GPUs with PCIe 3.0. The entire experiments take around 21,114 GPU hours in total.

Task. We select three kinds of popular Transformer workloads to evaluate the efficiency. The first is the long range arena (LRA) benchmark [37], which is a systematic and unified benchmark designed for efficient attention with long input sequences. It consists of 5 sequence classification tasks: ListOps [27] (a task to parse structured long input and get the answer), Text (byte-level text classification using the IMDb reviews [25] dataset), Retrieval (byte-level document retrieval using the ACL Anthology Network [32] dataset), Image (image classification on sequences of pixels using the CIFAR-10 [20] dataset), and Pathfinder [18, 23] (a long-range spatial dependency challenge). Their sequence lengths are 2K, 4K, 4K, 1K, and 1K, respectively. We use the conventional Transformer [38] as the backbone model and apply these efficient Transformer techniques by replacing the original MSA module. The second kind of workload is the NLP task. We take a popular RoBERTa [24] model as the backbone and perform masked language modeling [13] (MLM) pre-training task over WikiText-103 [26] dataset. To evaluate the model performance, we further fine-tune the pre-trained model on four text classification tasks in GLUE benchmark [40], including MNLI [42], QNLI [34], QQP [16] and SST-2 [35]. The sequence length is 512. The third workload is the CV task. Similarly, we take ViT [14] as the backbone and adopt SimMIM [43] to perform masked image modeling pre-training. We use Tiny-ImageNet-200 dataset [21] as the training set and fine-tune on CIFAR-10, CIFAR-100 [20] and Tiny-ImageNet-200 datasets to evaluate the classification accuracy. The sequence length is 256 (slightly shorter than NLP task due to the image resolution restriction). More details about each task's setups are listed in our Appendix. We repeat the experiments three times with different random seeds, and report averaged results (e.g., test accuracy) for each task.

Training Setting. When evaluating Transformers, it is common to take a pooling step to aggregate the information behind the output of all tokens from the sequence. CLS pooling is the most popular method that adds a special CLS token at the beginning of a sequence as the representation of the entire sequence after aggregation. The output hidden vector of the CLS token will be fed into the following classifier for classification, or other purposes. We also use CLS pooling in most of our experiments (i.e., CV, NLP) by default. For LRA tasks, we provide an additional MEAN pooling (i.e., averages all tokens) evaluation as suggested by [37].

Implementation. We re-implement these evaluations in PyTorch [22, 29] and our implementation is built on top of many existing open sourced projects [1–7]. The involved dense algebras are based on PyTorch's default `MatMul` operators and we also write an efficient block-sparse GPU kernels to support sparse algebras. Our code is also open sourced [8].

Configuration Setting. Different efficient attention methods may have distinct individual configurations, such as bucket size,

projection dimension, and so on. To conduct a fair comparison, it is necessary for us to limit all approaches within the same theoretical computation budget. To achieve this goal, we propose to use the `MatMul` phase's complexity as the measurement since it almost dominates the overall quadratic complexity with respect to input sequence length n. Specifically, we select the configuration-free approach, Cosformer's $O(4d^2n)$ as the maximum time complexity, and the configurations of the other approaches should be tuned to make their complexities as consistent as possible. Note that, our setting is not strictly fair because the system implementation may not be optimal, thus leading to inconsistent GPU execution time. For example, the theoretical complexity of positional selection methods is proportional to the sparsity of patterns, but it is hard to reach even if we implement them with the most efficient block-sparse GPU kernel since some additional overheads are inevitable. Therefore, it is important to carefully consider these impacts when setting the configurations.

3.2 Statistical Efficiency Evaluation (Q1)

3.2.1 ***LRA Benchmark.*** We first evaluate the statistical efficiency of all efficient attention on LRA benchmark datasets with rather long (i.e., ≥1K) input sequences.

Results Analysis. We train these efficient attention models from scratch and report the final test accuracy after model convergence in Table 2 and 3. We found that the positional selection methods generally outperform other efficient competitors when using CLS pooling. Compared with MEAN pooling, Longformer, and BigBird have remarkable improvements in terms of averaged accuracy in the ranking (i.e., $(4, 6) \rightarrow (1, 2)$). This is because their manually designed sparse patterns consider the global pattern in the beginning segment of each sequence that exactly covers the CLS token position thus carrying more valuable information. SparseTrans also considers global patterns but they are spread universally in each block which makes it better in MEAN pooling as it averages vectors from all positions.

The next three approaches, including Reformer, ClusterAttn, and LinearAttn, are not performing well on average for both pooling conditions. For the two contextual compression methods, we can even observe some significant model degradation cases compared with the full attention, like Reformer (CLS) on Text (4.7% ↓) and ClusterAttn (CLS) on Pathfinder (18.5% ↓). A possible reason is that the clustering or hashing approximation algorithms are losing too much information for long sequence tasks. While LinearAttn may over-simplify the `softmax` calculation leading to poor performance. For the other two activation kernelization methods, Cosformer performs close to full attention, and Performer even achieves the highest averaged accuracy in the MEAN pooling setting. It implies that carefully designed kernel functions and auxiliary optimization techniques could help to improve the long sequence modeling ability.

Linformer ranks last but one in averaged accuracy for both pooling settings. We infer that its linear projections (e.g., from $n = 1K$ to $p = 64$) might be too aggressive due to the computation budget constraints. Nyströmformer also suffers performance degradation, but still outperforms Linformer by a large margin. Synthesizer has the worst performance in our evaluations, which implies that the learnable input-agnostic random matrices might not be able to replace the conventional attention mechanism. TransNormer serves as a promotion for LinearAttn but exhibits worse performance in Text and Pathfinder tasks. This shows the refinement of TransNormer against typical activation kernelization methods works conditionally. In these two tasks, activation kernelization methods might perform better than positional selection methods, especially when using MEAN pooling. This can partly explain why block-local attention in TransNormer has a negative impact on model performance.

3.2.2 ***NLP and CV Tasks.*** We also evaluate popular real-world datasets including both NLP and CV workloads. For each workload, we first pre-train the model and then fine-tune the pre-trained model on some downstream tasks. For NLP tasks, we evaluate both validation perplexity (ppl, the lower the better) on the pre-trained MLM task and test accuracy on four GLUE tasks and report the average results in Table 4. For CV tasks, we report the averaged test accuracy on three popular image classification datasets in Table 5.

Results Analysis. As we can see, the positional selection methods have comparable or even better model performance (i.e., lower ppl or higher accuracy) than full attention, outdistancing most of the other methods on both NLP and CV tasks. An interesting finding is that they can even outperform full attention on NLP tasks. It might be because the sparse patterns not only preserve significant positions but also filter out meaningless attention values. In this way, the sparsification strengthens the most valuable information and improves the final model performance. However, on CV tasks, since images have a two-dimensional spatial structure and longer range dependencies, manually designed sparse patterns may not be optimal, leading to a slight performance loss. Here SparseTrans performs better than BigBird and Longformer because its block-global pattern enables queries to attend to keys from all blocks and helps to handle long-range dependencies.

The contextual compression methods have the potential to automatically capture complex sparse patterns. The CV tasks' results verify the ability to a certain extent. For example, ClusterAttn achieves the second highest average accuracy (the same as full attention) among all efficient attention methods. However, we also notice that it performs weaker on NLP tasks. In general, contextual compression methods can be competitive when the hashing and/or clustering produce accurate results, but also bring the risks of approximation's failure.

All three activation kernelization methods do not perform well in NLP or CV tasks. The removal of `softmax` produces a smoother attention distribution that dilutes the distinction between tokens and leads to degraded performance [31]. Although Cosformer and Performer propose novel optimizations on the kernel functions and achieves improvements compared with LinearAttn, they are still falling behind the positional selection and contextual compression counterparts.

For the low-rank factorization methods, Linformer still does not perform well on NLP and CV tasks due to the low dimensional projection and layer-wise sharing as discussed in Section 3.2.1. Surprisingly, Nyströmformer shows much better results, especially for NLP MLM pre-training and CV tasks. We also notice that its results on GLUE tasks are not as good as MLM. This inconsistency

Table 4: Results of NLP tasks

Model	**MLM** (ppl)↓	**MNLI**↑	**QNLI**↑	**QQP**↑	**SST-2**↑	**Avg.**↑
Full	5.62 6	68.61	80.43	81.66	83.14	78.46 5
Longformer	5.32 3	69.43	82.17	82.71	85.44	79.94 1
Bigbird	5.26 1	69.53	81.80	82.76	84.75	79.71 3
SparseTrans	5.40 4	69.75	81.24	81.88	85.89	79.69 4
Reformer	6.25 7	71.37	82.23	83.43	82.22	79.81 2
ClusterAttn	7.02 8	67.87	72.95	80.05	83.37	76.06 9
LinearAttn	11.03 13	62.05	64.21	76.82	82.34	71.35 12
Cosformer	7.79 9	65.08	69.17	79.83	83.60	74.42 11
Performer	8.55 10	65.51	71.41	79.46	82.68	74.76 10
Linformer	8.98 11	67.29	77.59	79.77	82.34	76.75 7
Nyströmformer	5.30 2	68.74	75.49	79.30	83.94	76.87 6
Synthesizer	9.18 12	59.31	66.37	76.09	82.80	71.14 13
TransNormer	5.60 5	68.47	72.41	79.30	82.51	76.35 8

Table 5: Results of CV tasks

Model	**CIFAR10**↑	**CIFAR100**↑	**Tiny-IN200**↑	**Avg.**↑
Full	93.06	72.65	54.87	73.53 2
Longformer	92.92	72.01	54.16	73.03 6
BigBird	92.54	72.50	54.21	73.08 5
SparseTrans	92.91	72.95	54.63	73.50 4
Reformer	92.24	70.95	52.77	71.99 8
ClusterAttn	93.20	72.79	54.59	73.53 2
LinearAttn	89.16	66.03	49.21	68.13 12
Cosformer	90.45	68.61	50.31	69.79 10
Performer	91.59	68.04	50.73	70.12 9
Linformer	90.87	66.84	49.33	69.01 11
Nyströmformer	93.31	72.79	56.36	74.15 1
Synthesizer	84.22	57.18	39.96	60.45 13
TransNormer	91.57	71.42	55.21	72.73 7

might be because the sequence lengths of original input data in GLUE datasets vary a lot, and the short sequences are set to be as long as the other sequences through padding. As a result, only a small part of landmarks in Nyströmformer are actually used for attention calculation, which leads to comparably worse model performance.

Similar to the results on LRA, Synthesizer still has the worst performance in almost all NLP and CV tasks. TransNormer successfully overcomes the weakness of activation kernelization by leveraging sparse patterns. In CV tasks where far token dependencies are more important, TransNormer catches up with positional selection methods, even though it only has simple block-local attention.

3.2.3 ***Interpretability Study.*** To better understand the fundamental differences of these efficient Transformer approaches, we study their impacts on the attention calculation results. We pick up one attention head of the last layer for each well-trained model on the NLP MLM task and visualize their attention matrices in Figure 3. Illustrations for more heads are in the Appendix.

As we can see, the attention score of full attention 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. Positional selection 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 contextual compression methods, their illustrated local and long-range dependencies change significantly, showing 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 blocks in the attention matrices. Activation 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. We also observe some local biases, especially for Cosformer and Performer due to their additional optimizations (e.g., re-weighting).

Table 6: MLM validation ppl for different sequence lengths

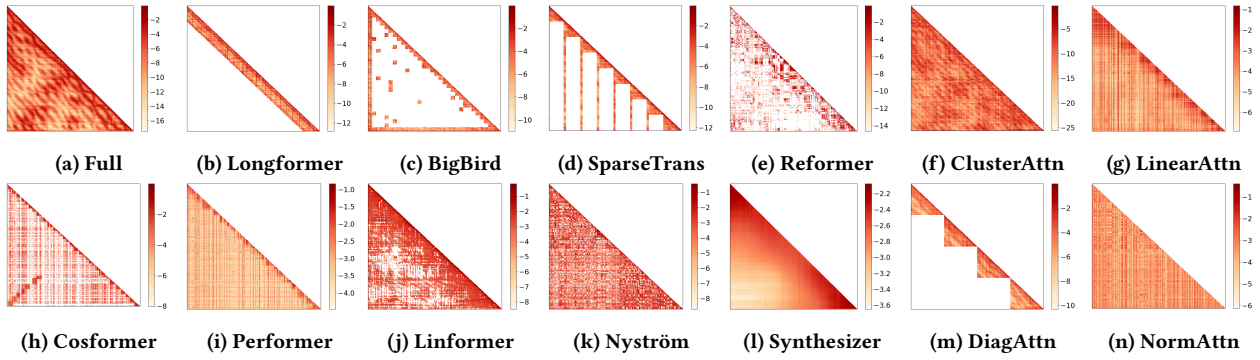
	$n = 512$			$n = 2048$		
Training Process	20%	60%	100%	20%	60%	100%
Full	12.85	6.04	5.62	9.25	5.31	5.00
Longformer	7.49	5.56	5.32	7.06	5.40	5.15
BigBird	6.97	5.46	5.26	6.84	5.39	5.14
SparseTrans	6.90	5.58	5.40	6.86	5.53	5.27
Reformer	8.56	6.54	6.25	9.18	7.13	6.84
ClusterAttn	53.06	7.51	6.97	51.03	29.70	26.16
LinearAttn	70.83	14.36	11.03	64.52	38.05	29.19
Cosformer	16.53	8.35	7.79	48.28	19.72	16.25
Performer	43.83	9.56	8.55	52.91	21.21	17.24
Linformer	71.25	11.86	8.98	56.84	33.25	29.84
Nyströmformer	6.61	5.47	5.30	6.32	5.35	5.21
Synthesizer	40.24	9.93	9.18	74.33	32.16	15.03
TransNormer	7.25	5.83	5.60	7.04	5.73	5.50

For low-rank factorization methods, we use the computed matrices which are used to take weighted averages of values as their attention matrices. The results involve some negative values, which are represented as the blank positions in Figures 3j and 3k. 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 activation kernelization methods do.

3.3 Impacts of Sequence Length (Q2)

To study the impacts of the input data sequence length, we take NLP MLM pre-training task as an example and dive deeper into the convergence process under the sequence length of 512 and 2048 respectively. We keep the number of processed tokens identical even if the sequence length varies. We set 250 epochs (each epoch means to process all tokens in the training dataset one pass) to be the 100% of the training progress. Other hyper-parameters and configurations are kept unchanged.

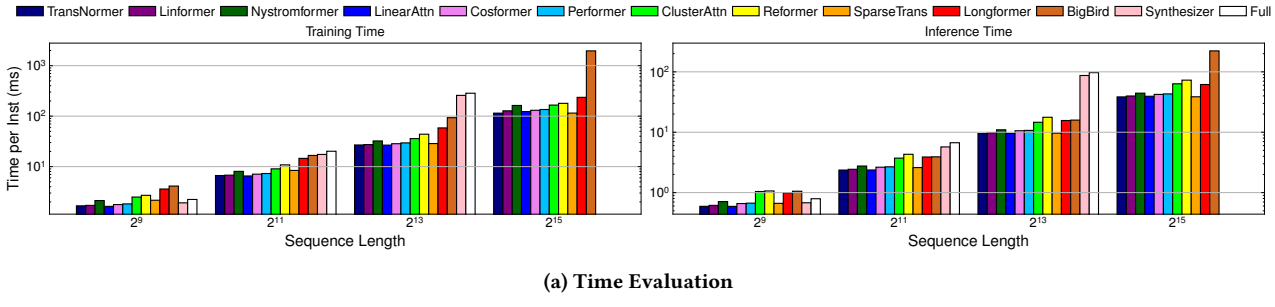
Results Analysis. As shown in Table 6, full attention improves a lot by increasing the sequence length. It demonstrates that making use of long-term contextual information results in more accurate guesses for masked tokens and better model performance. A similar trend could be observed in the positional selection methods and TransNormer. It further verifies the effectiveness of sparse



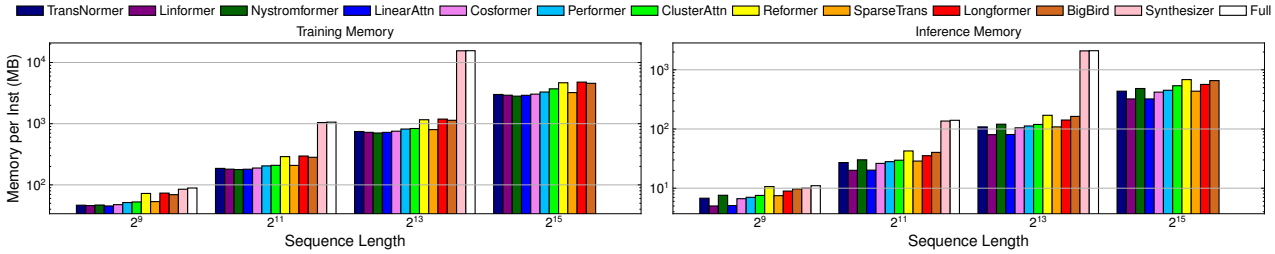
(a) Full (b) Longformer (c) BigBird (d) SparseTrans (e) Reformer (f) ClusterAttn (g) LinearAttn

(h) Cosformer (i) Performer (j) Linformer (k) Nyström (l) Synthesizer (m) DiagAttn (n) NormAttn

Figure 3: Visualization of attention matrices (after `softmax`) for all efficient attention methods (in log-scale for better illustration).



(a) Time Evaluation



(b) Memory Evaluation

Figure 4: Efficiency evaluation on different sequence lengths

patterns when the sequence gets longer. Contextual compression methods encounter performance decline when sequence length increases to 2048 since longer sequences increase the difficulty of hashing/clustering, and the inaccurate buckets/clusters make the model hard to quickly converge. We also provide the full convergence curves during training in the Appendix. Activation kernelization methods also suffer since the long sequences worsen the diluted attention distribution problem. The increasing sequence length also makes the performance of Linformer and Synthesizer drop as expected. Another interesting finding is that Nyströmformer has slight improvements thanks to the Nyström-based algorithm.

3.4 Computational and Memory Efficiency Evaluation (Q3)

In this section, we compare computational and memory efficiencies among efficient attention methods on different input sequence lengths. We use RoBERTa as the backbone and the hyper-parameters are the same as the ones in NLP tasks in Section 3.2.2. Note that, the theoretical computation budget limitation still holds in this experiment. Synthesizer and Vanilla occur the out-of-memory issue when $n = 2^{15}$ so they have empty bars for that case in Figure 4.

3.4.1 ***Computational Efficiency Evaluation.*** We report per instance execution time (ms) for training and inference procedure, as illustrated in Figure 4a. As we can see, low-rank factorization methods and activation kernelization methods along with TransNormer, have the fastest speed. Since LinearAttn only takes half of the maximum computation budget, it outperforms almost all the other competitors. Cosformer and Performer are slower than Linformer for different reasons. Though Linformer has additional MM Ops in the preprocessing phase, Cosformer has more MR and EA Ops to do re-weighting, and Performer has an additional feature mapping over all queries and keys, which takes some time. Nyströmformer is slower due to its iterative algorithm for the approximation of the

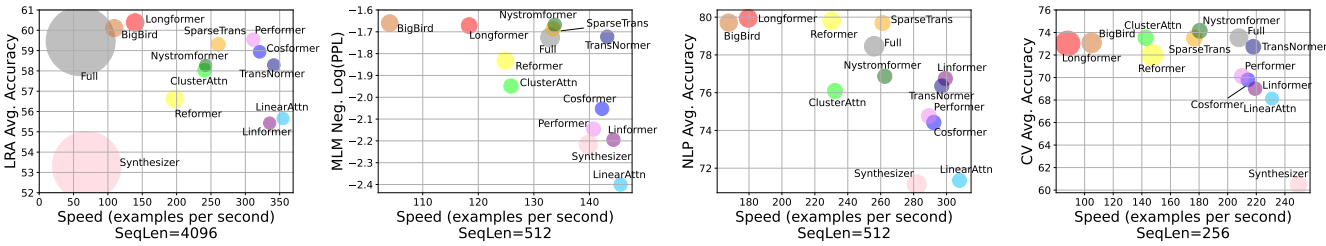


Figure 5: Performance ($y-$axis), speed ($x-$axis), and memory footprint (size of the circles) of different models in different tasks

Moore-Penrose pseudoinverse. TransNormer is fast since its DiagAttention uses the simple block-local pattern, which has an easy and highly efficient implementation, and its NormAttention saves time by canceling computing denominators for normalization.

Positional selection and contextual compression methods (except SparseTrans) have worse performance on time efficiency. This is largely because they require more MR Ops, and perform more complex transformations on tensors, which are not friendly to GPUs with massive parallel threads. To implement the attention with sparse patterns efficiently, positional selection methods tend to re-arrange query/key/value matrices and compute attention score matrix and context values multiple times to get results of different basic patterns, then combine all of them. Such redundant calculations lead to much higher practical time complexity compared with the theoretical one. In our evaluation, we follow HuggingFace's BigBird implementation, which divides the computation of attention into five parts, thus involving a large amount of MR and MM Ops. These operators have a serious negative impact on the efficiency of the model. The impact will get larger when we fix the block size b and increases the sequence length n, which increases the number of blocks, resulting in further reductions in the parallelism. SparseTrans is an exception because the block-local and block-global attention can be directly implemented by chunk-level matrix multiplication, without introducing redundant calculation. When n is over 4096, only the block-local pattern is used for aligning with the compute budget, which further simplifies its calculation.

Reformer and ClusterAttn are slower than Longformer in inference. This is most likely attributable to the preprocessing phase, in which Reformer and ClusterAttn consume a lot of time on hashing, clustering, and sorting. However, the situation is reversed in training, because some procedures do not need gradient calculations in the backward process, and some of them have implemented customized efficient kernels, such as some sorting and broadcasting functions for clustering.

Synthesizer still has quadratic complexity to sequence length n. However, the complexity of Synthesizer in `MatMul` phase is $(p+d)n^2$ as shown in Table 1, which is smaller than that of Vanilla ($2dn^2$) when $p < d$. Therefore, the curve of Synthesizer is approaching the one of Vanilla as n increases, but the gap will not be eliminated. It is also worth noting that when $n \leq 512$, most of the positional selection and contextual compression methods are slower than full attention, while Synthesizer is faster even though maintains quadratic complexity.

3.4.2 ***Memory Efficiency Evaluation.*** We then study the per-sequence peak runtime GPU memory usage (MB) for training and inference procedure, as illustrated in Figure 4b.

Low-rank factorization methods and activation kernelization methods have comparatively low memory usage. Linformer uses even less memory than LinearAttn in inference, because the feature mapping of LinearAttn adds a non-trivial amount of memory usage, substituting the softmax operation in Linformer. Unlike the case in computational efficiency evaluation, TransNormer has larger memory usage than Linformer. It is mainly due to the generated blocked attention mask in DiagAttention.

Positional selection and contextual compression methods (except SparseTrans) do not perform well on memory efficiency either. Specific attention masks generated for different sparse patterns consume memory. More memory-intensive operations are performed in positional selection methods, while memory-consuming preprocessing phases are employed in contextual compression methods. They both suffer from redundant data movement operations in attention computation. Besides, when n is small, they still get better efficiency compared to Vanilla. Reformer and BigBird use many (non-in-place) MR Ops. Longformer computes global attention with additional global queries/keys/values. Therefore they have larger memory usage in both training and inference. Synthesizer has a quadratic memory complexity and still consumes the full attention matrix's memory footprint.

3.5 Overall Discussion (Q4)

We then discuss the overall trade-offs for each individual task in Figure 5. In each figure, the $y-$axis represents the corresponding task statistical efficiency (in different evaluation metrics) and the $x-$axis shows the computation speed (i.e., the number of processed sequences per second for training). All evaluated approaches are expressed as circles with different colors and each circle's center coordinates refer to its corresponding statistical and computational efficiencies. The size of these circles illustrates the memory efficiency (i.e., memory footprint) of each approach.

We found that, in the LRA benchmark with long sequence, full attention has unbearable time and memory overheads, but achieves superior model performance. The kernelization methods (e.g., Performer and Cosformer) get the best balance between performance and speed with low memory usage. Positional selection methods (e.g., Longformer and BigBird) are also good alternatives to full attention with at least 1.81× speed up and 8.07× memory saving. But these methods are slower than full attention in NLP tasks with a

slightly shorter sequence length of 512 in practice except for SparseTrans. For MLM pre-training tasks, full attention is better than all contextual compression methods in terms of both performance and speed. TransNormer still achieves a good balance in this case. Only SparseTrans surpasses full attention in all efficiency metrics. Other positional selection methods are slower, and the rest categories of methods have obvious performance degradation. In CV tasks with much shorter sequences, full attention itself is quite a good solution, as most efficient attention mechanisms are either slower or weaker. Due to the combination of positional selection and activation kernelization, TransNormer could be an alternative solution for less memory usage and faster speed with slight performance loss.

REFERENCES

[1] 2021. Fast Transformers. https://github.com/idiap/fast-transformers.
[2] 2022. CosFormer. https://github.com/OpenNLPLab/cosFormer.
[3] 2022. Long-Range Arena Benckmark. https://github.com/google-research/long-range-arena.
[4] 2022. Nyströmformer. https://github.com/mlpen/Nystromformer.
[5] 2022. Performer Pytorch. https://github.com/lucidrains/performer-pytorch.
[6] 2022. Transnormer. https://github.com/OpenNLPLab/Transnormer.
[7] 2023. Huggingface Transformers. https://github.com/huggingface/transformers.
[8] 2023. X-former Elucidator. https://github.com/Hsword/X-former-Elucidator.
[9] Iz Beltagy, Matthew E Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150* (2020).
[10] Shiyang Chen, Shaoyi Huang, Santosh Pandey, Bingbing Li, Guang R Gao, Long Zheng, Caiwen Ding, and Hang Liu. 2021. Et: re-thinking self-attention for transformer models on gpus. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–18.
[11] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. 2019. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509* (2019).
[12] Krzysztof Marcin Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamás Sarlós, Peter Hawkins, Jared Quincy Davis, Afroz Mohiuddin, Lukasz Kaiser, David Benjamin Belanger, Lucy J. Colwell, and Adrian Weller. 2021. Rethinking Attention with Performers. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net. https://openreview.net/forum?id=Ua6zuk0WRH
[13] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL-HLT*. 4171–4186. https://doi.org/10.18653/v1/n19-1423
[14] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*.
[15] Andrei Ivanov, Nikoli Dryden, Tal Ben-Nun, Shigang Li, and Torsten Hoefler. 2021. Data movement is all you need: A case study on optimizing transformers. *Proceedings of Machine Learning and Systems* 3 (2021), 711–732.
[16] Shankar Iyer, Nikhil Dandekar, and Kornl Csernai. 2016. First quora dataset release: Question pairs. (2016). https://data.quora.com/First-Quora-Dataset-Release-Question-Pairs
[17] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. 2020. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International Conference on Machine Learning*. PMLR, 5156–5165.
[18] Junkyung Kim, Drew Linsley, Kalpit Thakkar, and Thomas Serre. 2020. Disentangling neural mechanisms for perceptual grouping. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net. https://openreview.net/forum?id=HJxrVA4FDS
[19] Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. 2020. Reformer: The Efficient Transformer. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net. https://openreview.net/forum?id=rkgNKkHtvB
[20] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. (2009).
[21] Ya Le and Xuan Yang. 2015. Tiny imagenet visual recognition challenge. *CS 231N* 7, 7 (2015), 3.
[22] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, and Soumith Chintala. 2020. PyTorch Distributed: Experiences on Accelerating Data Parallel Training. *PVLDB* 13, 12 (2020), 3005–3018.
[23] Drew Linsley, Junkyung Kim, Vijay Veerabadran, Charles Windolf, and Thomas Serre. 2018. Learning long-range spatial dependencies with horizontal gated recurrent units. In *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Eds.), Vol. 31. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2018/file/ec8956637a99787bd197eacd77acce5e-Paper.pdf
[24] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *CoRR* abs/1907.11692 (2019). arXiv:1907.11692
[25] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. 2011. Learning Word Vectors for Sentiment Analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, Portland, Oregon, USA, 142–150. https://aclanthology.org/P11-1015
[26] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2017. Pointer Sentinel Mixture Models. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net. https://openreview.net/forum?id=Byj72udxe

[27] Nikita Nangia and Samuel Bowman. 2018. ListOps: A Diagnostic Dataset for Latent Tree Learning. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Student Research Workshop*. Association for Computational Linguistics, New Orleans, Louisiana, USA, 92–99. https://doi.org/10.18653/v1/N18-4013

[28] Xiaonan Nie, Xupeng Miao, Zhi Yang, and Bin Cui. 2022. TSPLIT: Fine-grained GPU Memory Management for Efficient DNN Training via Tensor Splitting. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, 2615–2628.

[29] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32 (2019).

[30] Zhen Qin, XiaoDong Han, Weixuan Sun, Dongxu Li, Lingpeng Kong, Nick Barnes, and Yiran Zhong. 2022. The Devil in Linear Transformer. *arXiv preprint arXiv:2210.10340* (2022).

[31] Zhen Qin, XiaoDong Han, Weixuan Sun, Dongxu Li, Lingpeng Kong, Nick Barnes, and Yiran Zhong. 2022. The Devil in Linear Transformer. *arXiv preprint arXiv:2210.10340* (2022).

[32] Dragomir R. Radev, Pradeep Muthukrishnan, Vahed Qazvinian, and Amjad Abu-Jbara. 2013. The ACL Anthology Network Corpus. *Lang. Resour. Eval.* 47, 4 (dec 2013), 919–944. https://doi.org/10.1007/s10579-012-9211-2

[33] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 2020. ZeRO: memory optimizations toward training trillion parameter models. In *SC 2020*. IEEE/ACM, 20. https://doi.org/10.1109/SC41405.2020.00024

[34] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ Questions for Machine Comprehension of Text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Austin, Texas, 2383–2392. https://doi.org/10.18653/v1/D16-1264

[35] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Seattle, Washington, USA, 1631–1642. https://aclanthology.org/D13-1170

[36] Yi Tay, Dara Bahri, Donald Metzler, Da-Cheng Juan, Zhe Zhao, and Che Zheng. 2021. Synthesizer: Rethinking Self-Attention for Transformer Models. https://openreview.net/forum?id=H-SPvQtMwm

[37] Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. 2021. Long Range Arena : A Benchmark for Efficient Transformers. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net. https://openreview.net/forum?id=qVyeW-grC2k

[38] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *NeurIPS 2017*. 5998–6008.

[39] Apoorv Vyas, Angelos Katharopoulos, and François Fleuret. 2020. Fast transformers with clustered attention. *Advances in Neural Information Processing Systems* 33 (2020), 21665–21674.

[40] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019. GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net. https://openreview.net/forum?id=rJ4km2R5t7

[41] Sinong Wang, Belinda Z Li, Madian Khabsa, Han Fang, and Hao Ma. 2020. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768* (2020).

[42] Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. Association for Computational Linguistics, New Orleans, Louisiana, 1112–1122. https://doi.org/10.18653/v1/N18-1101

[43] Zhenda Xie, Zheng Zhang, Yue Cao, Yutong Lin, Jianmin Bao, Zhuliang Yao, Qi Dai, and Han Hu. 2022. SimMIM: A Simple Framework for Masked Image Modeling. In *International Conference on Computer Vision and Pattern Recognition (CVPR)*.

[44] Yunyang Xiong, Zhanpeng Zeng, Rudrasis Chakraborty, Mingxing Tan, Glenn Fung, Yin Li, and Vikas Singh. 2021. Nyströmformer: A Nyström-based Algorithm for Approximating Self-Attention. (2021).

[45] Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. 2020. Big bird: Transformers for longer sequences. *Advances in neural information processing systems* 33 (2020), 17283–17297.

[46] Ce Zhang and Christopher Ré. 2014. DimmWitted: A Study of Main-Memory Statistical Analytics. *PVLDB* 7, 12 (2014), 1283–1294.

[47] Q Zhen, W Sun, H Deng, D Li, Y Wei, B Lv, J Yan, L Kong, and Y Zhong. 2022. cosFormer: Rethinking Softmax In Attention. In *International Conference on Learning Representations*.

[48] Liyan Zheng, Haojie Wang, Jidong Zhai, Muyan Hu, Zixuan Ma, Tuowei Wang, Shizhi Tang, Lei Xie, Kezhao Huang, and Zhihao Jia. 2022. OLLIE: Derivation-based Tensor Program Optimizer. *arXiv preprint arXiv:2208.02025* (2022).

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 2 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 $O(\cdot)$ notation for simplicity.

Full. For a single head, full attention 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 attention methods have three linear projections on $\mathbf{X}$ to produce $\mathbf{Q}, \mathbf{K}, \mathbf{V}$, thus having the same complexity in linear phase as full attention. We will only mention the difference in linear phase for rest of the models. Full attention 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 full attention, 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 full attention, 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 an activation 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 $\mathbf{Q}(\mathbf{K}^\top\mathbf{V})$ in its sine and cosine variants respectively. 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 attention calculation in Linformer can be written as:

$$\mathbf{X} = \texttt{softmax}\left(\frac{\mathbf{Q}\,(\mathbf{P}_1\mathbf{K})^\top}{\sqrt{d}}\right)(\mathbf{P}_2\mathbf{V}), \tag{6}$$

where $\mathbf{P}_1, \mathbf{P}_2 \in \mathbb{R}^{p\times n}$ are low-dim projections. The calculation for $\mathbf{P}_1\mathbf{K}$ and $\mathbf{P}_2\mathbf{V}$ in E.q. (6) 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. The attention calculation in Nyströmformer-can be written as:

$$\begin{aligned}\tilde{\mathbf{X}} &= \texttt{softmax}\left(\frac{\mathbf{Q}\tilde{\mathbf{K}}^\top}{\sqrt{d}}\right)\left(\texttt{softmax}\left(\frac{\tilde{\mathbf{Q}}\tilde{\mathbf{K}}^\top}{\sqrt{d}}\right)\right)^+ \\ &\qquad\qquad \texttt{softmax}\left(\frac{\tilde{\mathbf{Q}}\mathbf{K}^\top}{\sqrt{d}}\right)\mathbf{V} \\ &= \mathbf{A}_k\mathbf{A}_{qk}^+\mathbf{A}_q\mathbf{V} = (\mathbf{A}_k\mathbf{A}_{qk}^+)(\mathbf{A}_q\mathbf{V}),\end{aligned} \tag{7}$$

here $\tilde{\mathbf{Q}}, \tilde{\mathbf{K}} \in \mathbb{R}^{p \times d}$ are low-dim sampling matrices. Nyströmformer consists of six matrix multiplications for attention calculation in E.q. (7). 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 attention calculation in Sythesizer can be written as:

$$X = \texttt{softmax}\left(\mathbf{R}_1\mathbf{R}_2^\top\right)\mathbf{V}, \tag{8}$$

here $\mathbf{R}_1, \mathbf{R}_2 \in \mathbb{R}^{n \times p}$ are low-dim learnable matrices. 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. (8) 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 positional selection 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 an activation kernelization method. It applies feature map on $\mathbf{Q}, \mathbf{K}$ in preprocessing phase and follows the calculation scheme of activation kernelization methods, which costs $2d^2n$ in `MatMul` phase.

B EXPERIMENTAL SETUP DETAILS

LRA. We use the conventional Transformer with softmax attention as the baseline, which is also termed as full attention (or Full for simplicity) in this work. Every efficient method is implemented on the same Transformer backbone, except that we replace the 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 attention methods. 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.

C MORE EXPERIMENT RESULTS

C.1 Pre-training Tasks

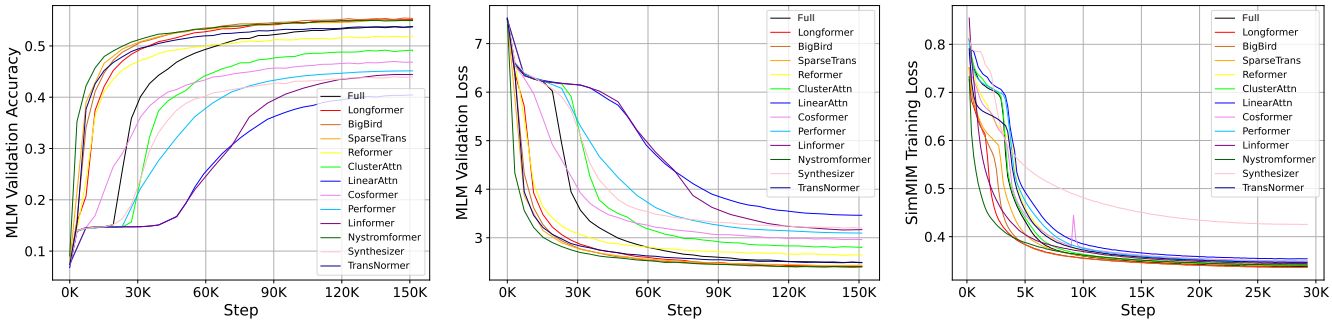
For MLM pre-training task, we illustrate the curves of accuracy and loss on the validation set for all efficient attention methods along with the full attention, as shown in Figures 6a and 6b respectively. For SimMIM pre-training task, we use the whole Tiny-ImageNet-200 dataset for training, thus illustrating training loss curves in Figure 6c.

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 attention methods as well as the full attention are shown in Figure 7.

D MORE ATTENTION VISUALIZATION RESULTS

We provide the attention visualization results of different Transformer variants for all 4 heads and 4 layers in Figures 8-20. The attention values are scaled logarithmically for better illustration.



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

Figure 6: Results on pre-training tasks

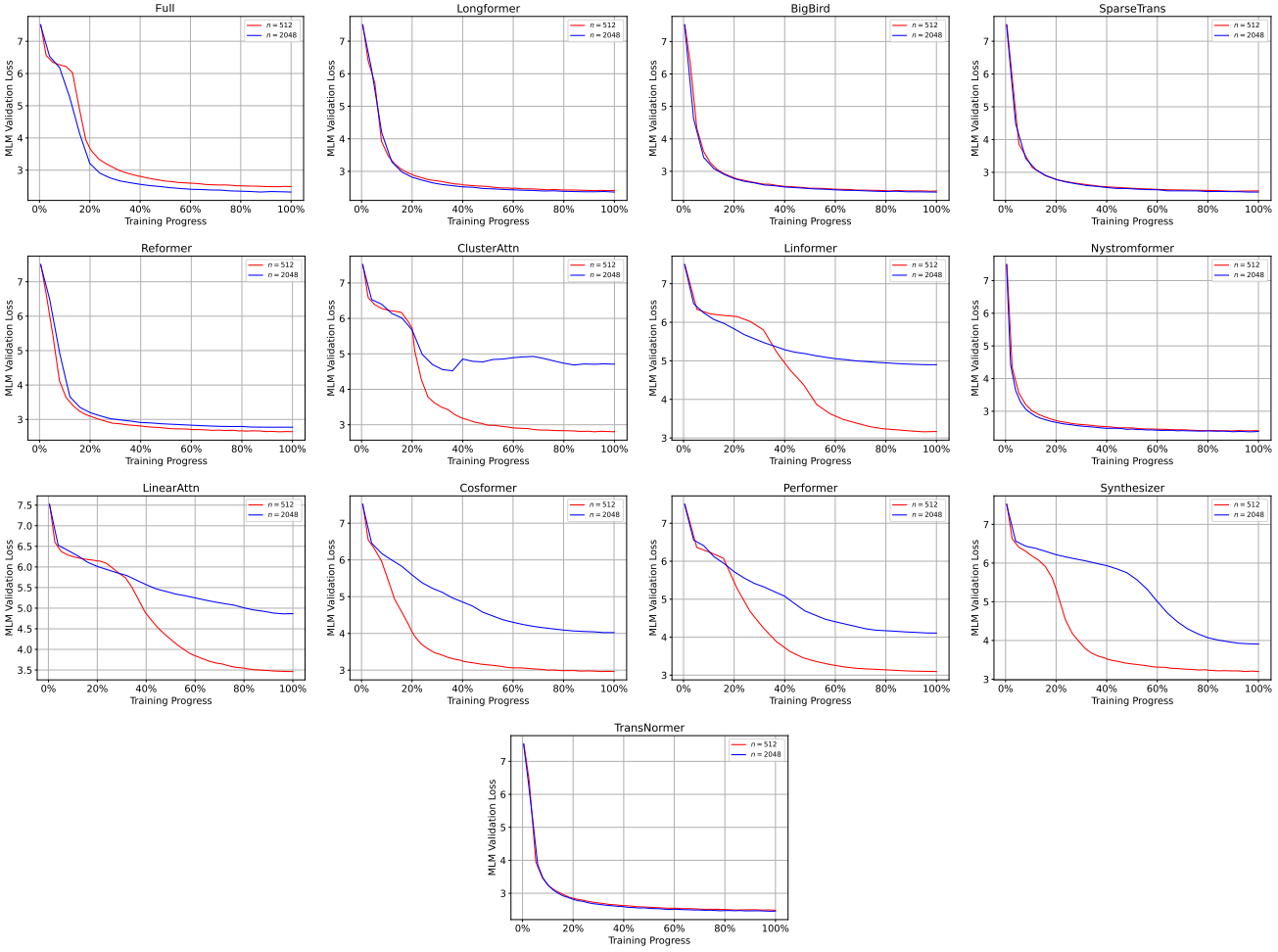
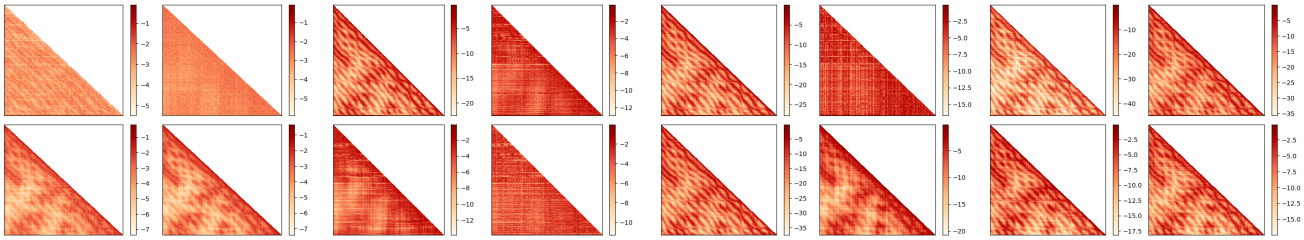
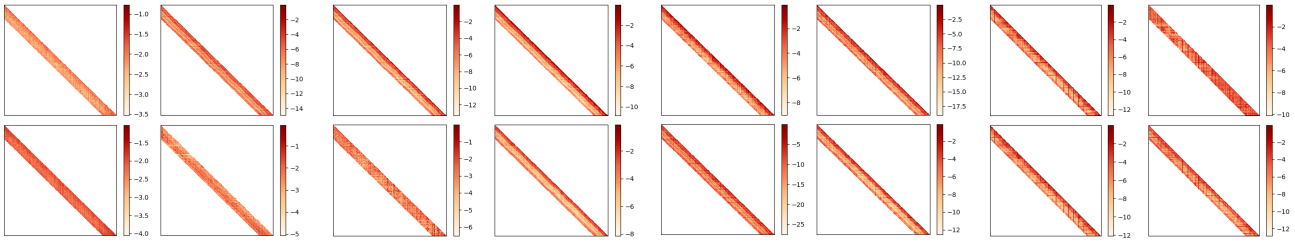


Figure 7: MLM validation loss on different sequence lengths for efficient attention methods



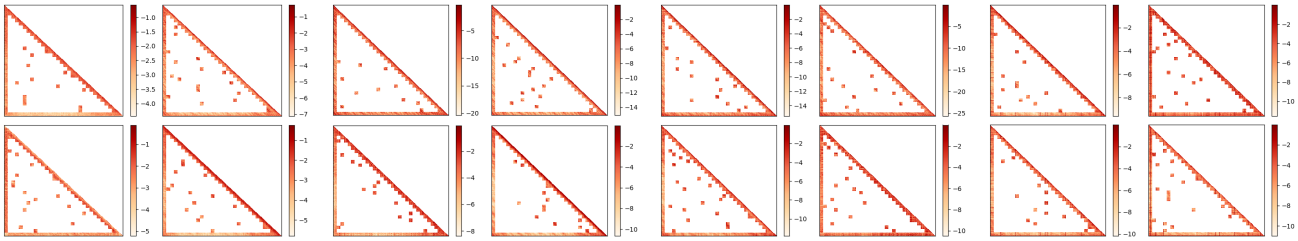
(a) Layer 1 (b) Layer 2 (c) Layer 3 (d) Layer 4

Figure 8: Visualization of attention matrices for all attention heads in Full Attention



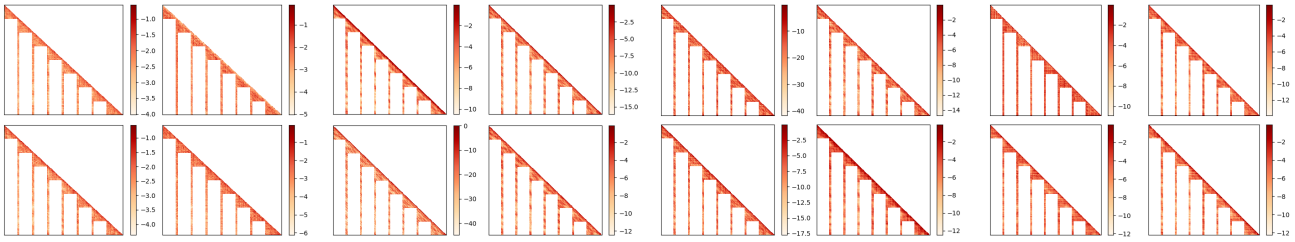
(a) Layer 1 (b) Layer 2 (c) Layer 3 (d) Layer 4

Figure 9: Visualization of attention matrices for all attention heads in Longformer



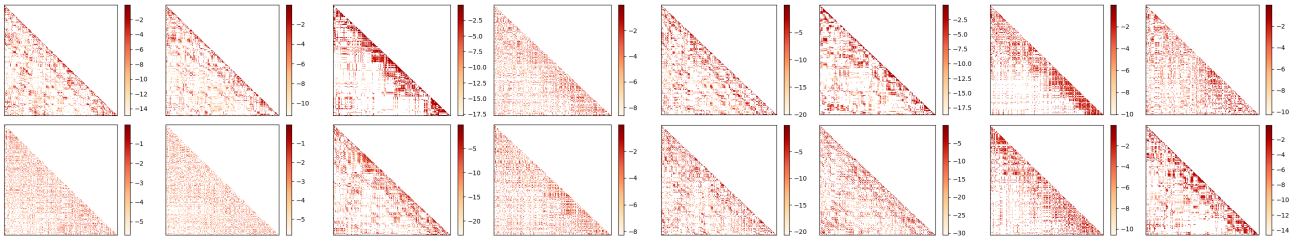
(a) Layer 1 (b) Layer 2 (c) Layer 3 (d) Layer 4

Figure 10: Visualization of attention matrices for all attention heads in BigBird



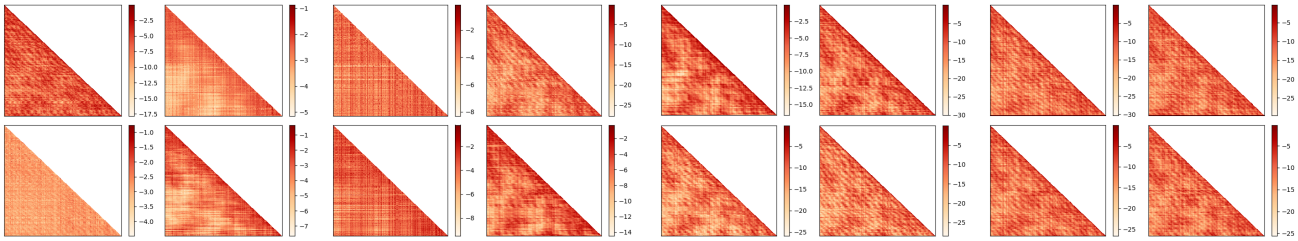
(a) Layer 1 (b) Layer 2 (c) Layer 3 (d) Layer 4

Figure 11: Visualization of attention matrices for all attention heads in SparseTrans



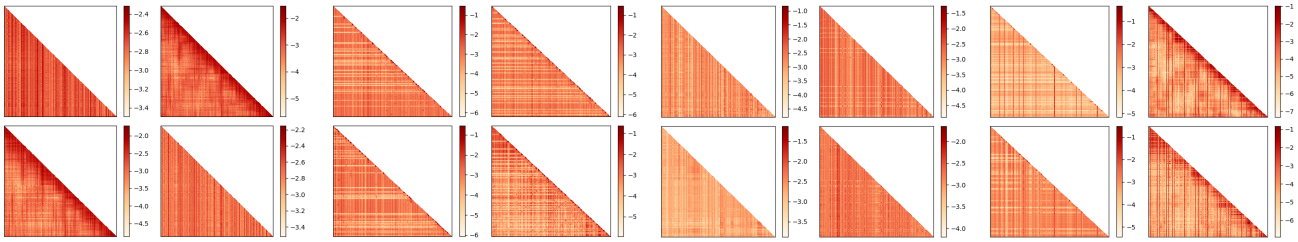
(a) Layer 1 (b) Layer 2 (c) Layer 3 (d) Layer 4

Figure 12: Visualization of attention matrices for all attention heads in Reformer



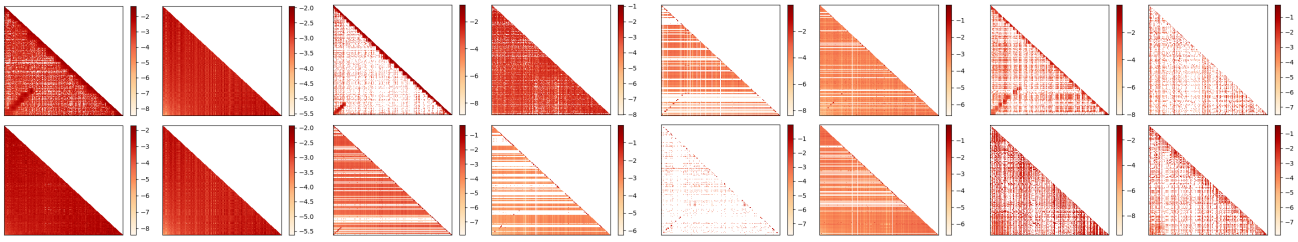
(a) Layer 1 (b) Layer 2 (c) Layer 3 (d) Layer 4

Figure 13: Visualization of attention matrices for all attention heads in ClusterAttn



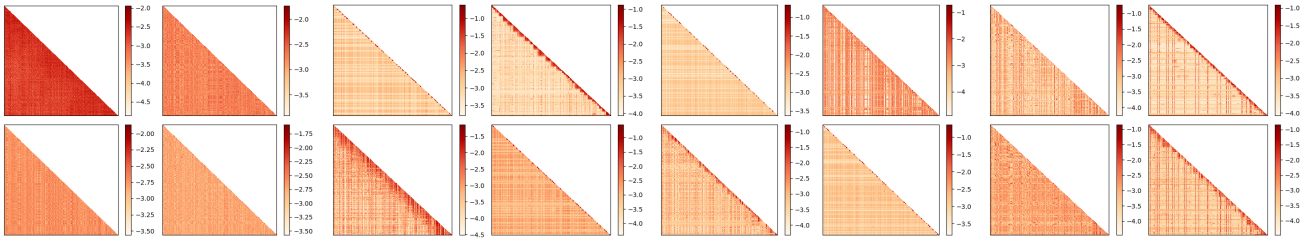
(a) Layer 1 (b) Layer 2 (c) Layer 3 (d) Layer 4

Figure 14: Visualization of attention matrices for all attention heads in LinearAttn



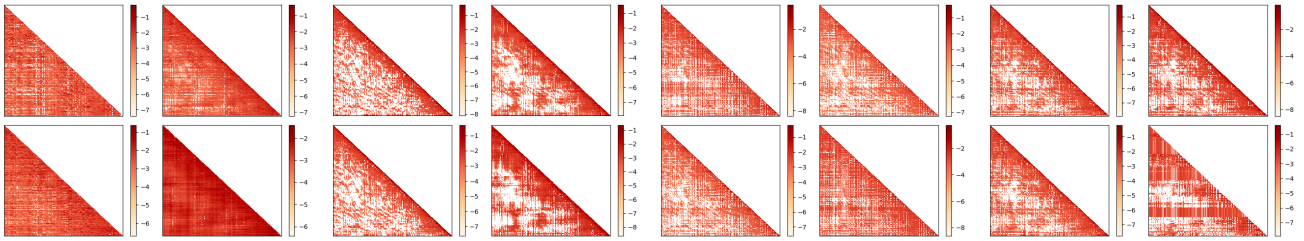
(a) Layer 1 (b) Layer 2 (c) Layer 3 (d) Layer 4

Figure 15: Visualization of attention matrices for all attention heads in Cosformer



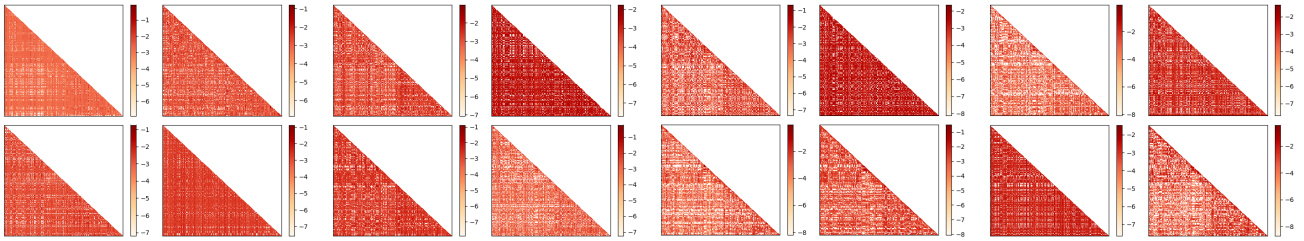
(a) Layer 1 (b) Layer 2 (c) Layer 3 (d) Layer 4

Figure 16: Visualization of attention matrices for all attention heads in Performer



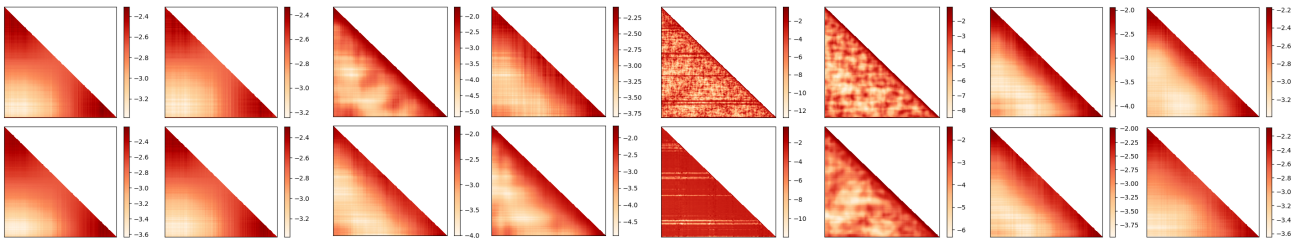
(a) Layer 1 (b) Layer 2 (c) Layer 3 (d) Layer 4

Figure 17: Visualization of attention matrices for all attention heads in Linformer



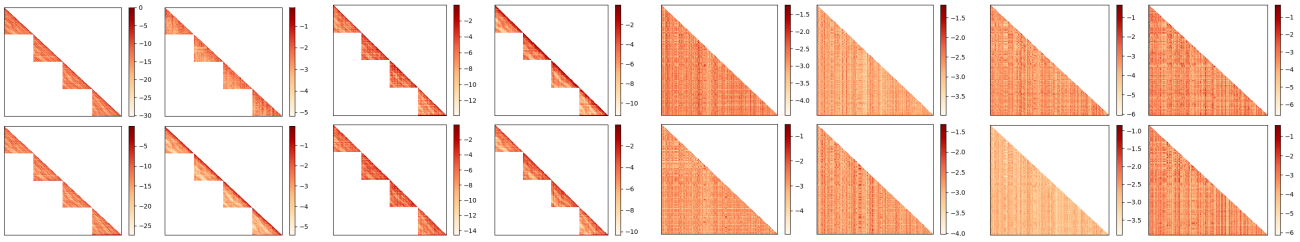
(a) Layer 1 (b) Layer 2 (c) Layer 3 (d) Layer 4

Figure 18: Visualization of attention matrices for all attention heads in Nyströmformer



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

Figure 19: Visualization of attention matrices for all attention heads in Synthesizer



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

Figure 20: Visualization of attention matrices for all attention heads in TransNormer