From Transformer to Sliceformer: Make Multi-Head Attention as Simple as Sorting

Anonymous Author(s)
Affiliation
Address
email

Abstract

As one of the most popular neural network modules, Transformer plays a central role in many fundamental deep learning models, e.g., the ViT in computer vision and the BERT and GPT in natural language processing. Although without strict theoretical supports, the effectiveness of the Transformer is often attributed to its multi-head attention (MHA) mechanism. In this study, we challenge this empirical opinion and discuss the limitations of the MHA mechanism, including the numerical issues caused by its softmax operation and the restricted capacity caused by the multi-head architecture. Considering the above issues and the recent development tendency of the attention layer, we propose an effective and efficient surrogate of the Transformer, called Sliceformer. Our Sliceformer replaces the MHA mechanism with an extremely simple "slicing-sorting" operation, i.e., projecting inputs linearly to a latent space and sorting them along different feature dimensions. We analyze the connections and differences between the proposed operation and the MHA mechanism and demonstrate the advantages of the corresponding Sliceformer on computational complexity, scalability, and numerical stability. Experiments in the Long-Range Arena benchmark, image classification, and molecular analysis show that our Sliceformer achieves comparable or better accuracy with lower memory cost, faster speed, and better numerical stability than the Transformer and its variants.

1 Introduction

Transformer [1] has been dominant in deep learning research for recent years. It works as a backbone module in many fundamental models, achieving outstanding performance in various application scenarios. Currently, the most successful language models like BERT [2] and GPT [3] are built based on the Transformer or its variants [4, 5], which outperforms classic recurrent neural network (RNN) architectures on both effectiveness and efficiency. In the field of computer vision, the Vision Transformers (ViTs) [6–8] have achieved better performance in many image recognition and understanding tasks compared to convolutional neural networks (CNNs). Recently, the Transformer-based models have been designed for the structured data in different applications, including the Informer [9] for time series broadcasting, the Graphormer [10] for molecular representation, the Set-Transformer [11] and Point-Transformer [12] for point cloud modeling, and so on. More and more cases show the tendency that the Transformer is becoming an indispensable choice when developing deep learning models.

Although without strict theoretical support, the effectiveness of the Transformer is often attributed to the multi-head attention (MHA) mechanism [1] behind it. This empirical but dominant opinion impacts the design and modification of the Transformer significantly, which, in our opinion, might have restricted the development of new model architectures to some degree. As shown in Table 1,

Submitted to 37th Conference on Neural Information Processing Systems (NeurIPS 2023). Do not distribute.

Table 1: A comparison for representative Transformers and our Sliceformer on their attention mechanisms. We show one attention head for each transformer, in which the input $\boldsymbol{X} \in \mathbb{R}^{N\times d}$, the value $\boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}_V \in \mathbb{R}^{N\times D}$, the query $\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}_Q \in \mathbb{R}^{N\times D}$, and the key $\boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}_K \in \mathbb{R}^{N\times D}$.

Model	Attention$(\boldsymbol{V}; \boldsymbol{Q}, \boldsymbol{K})$	Complexity	Attention Structure
Transformer [1]	Softmax $\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{D}}\right)\boldsymbol{V}$	$\mathcal{O}(DN^2)$	Dense + Row-stochastic
SparseTrans [4]	Local2D-Softmax $\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{D}}\right)\boldsymbol{V}$	$\mathcal{O}(DN^{1.5})$	Sparse + Row-stochastic
Longformer [13]	Local1D-Softmax $\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{D}}\right)\boldsymbol{V}$	$\mathcal{O}(DNL)$	Sparse + Row-stochastic
Reformer [14]	LSH-Softmax $\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{D}}\right)\boldsymbol{V}$	$\mathcal{O}(DN\log N)$	Sparse + Row-stochastic
CosFormer [15]	$(\boldsymbol{Q}_{\cos}\boldsymbol{K}_{\cos}^\top + \boldsymbol{Q}_{\sin}\boldsymbol{K}_{\sin}^\top)\boldsymbol{V}$	$\mathcal{O}(\min\{DE_{QK}, NE_Q\})$	Sparse
Performer [16]	$\phi_r(\boldsymbol{Q})\phi_r(\boldsymbol{K})^\top\boldsymbol{V}$	$\mathcal{O}(DNr)$	Low-rank
Linformer [17]	Softmax $\left(\frac{\boldsymbol{Q}\psi_r(\boldsymbol{K})^\top}{\sqrt{D}}\right)\psi_r(\boldsymbol{V})$	$\mathcal{O}(DNr)$	Low-rank + Row-stochastic
Sinkformer [18]	Sinkhorn$_K$ $\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{D}}\right)\boldsymbol{V}$	$\mathcal{O}(KDN^2)$	Dense + Doubly stochastic
Sliceformer	Sort$_{\text{col}}(\boldsymbol{V})$	$\mathcal{O}(DN\log N)$	Full-rank + Sparse + Doubly stochastic

[1] "Local1D" considers L local data in a sequence. "Local2D" considers the row-wise and column-wise local data for a sequence zigzagging in the 2D space. "LSH" denotes Locality-sensitive Hashing.
[2] $\phi_r : \mathbb{R}^D \mapsto \mathbb{R}^r$, and $\phi_r(\boldsymbol{Q}), \phi_r(\boldsymbol{K}) \in \mathbb{R}^{N\times r}$; $\psi_r : \mathbb{R}^N \mapsto \mathbb{R}^r$, and $\psi_r(\boldsymbol{K}), \psi_r(\boldsymbol{V}) \in \mathbb{R}^{r\times D}$.
[3] $\boldsymbol{K}_{\cos} = \text{diag}(\{\cos\frac{\pi i}{2M}\}_{i=1}^N)\text{ReLU}(\boldsymbol{K})$, $\boldsymbol{K}_{\sin} = \text{diag}(\{\sin\frac{\pi i}{2M}\}_{i=1}^N)\text{ReLU}(\boldsymbol{K})$. So are $\boldsymbol{Q}_{\cos}$ and $\boldsymbol{Q}_{\sin}$. E_{QK} is the number of nonzero elements in $\boldsymbol{Q}_{\cos}\boldsymbol{K}_{\cos}^\top$. E_Q is the number of nonzero elements in $\boldsymbol{Q}_{\cos}$.
[4] "Sinkhorn$_K$" means applying K-step Sinkhorn iterations.
[5] Note that, our Sliceformer does not need the "multi-head" architecture because of the simplicity of sorting.

many variants of Transformer have been proposed to i) improve the efficiency of MHA (e.g., designing sparse or low-rank attention maps [4, 14, 17]), ii) enhance the interpretability of MHA (e.g., revisiting attention maps through the lens of kernel theory [19, 15] and optimal transport [20, 18]), or iii) impose more side information on attention maps [21, 10, 22]. However, they still rely on the classic "query-key-value" architecture of MHA. Little attention is paid to studying the necessity of the MHA itself or, more ambitiously, replacing the MHA with a new surrogate. Some work replaces the MHA with some other architectures, e.g., RNN [23], MLP [24], and Structured State Space Sequential Model [25]. Still, they mainly focus on reusing existing neural networks in specific tasks rather than designing a new and general module.

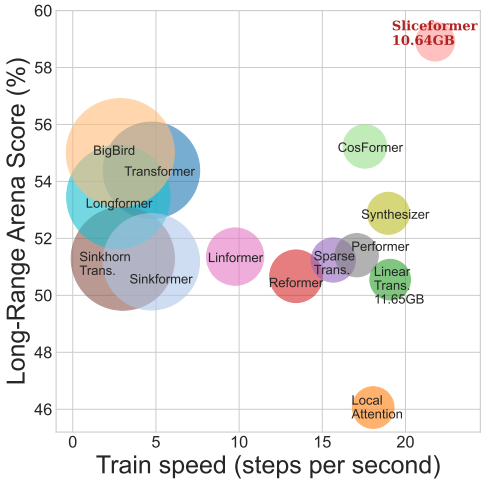


Figure 1: The comparison for various Transformers and our Sliceformer on the LRA benchmark. The length of the sequence is 3K. The x-axis corresponds to the number of training steps per second. The y-axis corresponds to the average score (%) on the LRA benchmark. The peak memory usage of each model is represented as the area of the corresponding circle. For a better comparison, the values (GB) of the top-2 models are shown.

In this study, we challenge the architecture of MHA, replacing it with an extremely simple "slicing-sorting" operation and developing a surrogate of the Transformer called Sliceformer. In particular, our work is motivated by the MHA's drawbacks and the attention map's possibly-desired structure. Firstly, we attribute the MHA's numerical issues to the softmax operation and point out the potential loss of model capacity caused by the multi-head structure. Secondly, the development tendency of the MHA and its variants implies that we shall pursue as many sparse and doubly stochastic attention maps as possible

for projected samples. Based on the analysis above, we propose the "slicing-sorting" operation, which projects samples linearly to a latent space and sorts them along different feature dimensions. Replacing the MHA mechanism of the Transformer with the slicing-sorting operation leads to the proposed Sliceformer.

As shown in Table 1, our slicing-sorting operation only preserves the linear map from the input $\boldsymbol{X}$ to the value matrix $\boldsymbol{V}$. In addition, we do not need the "multi-head" structure because concatenating different linear maps is equivalent to increasing the columns of $\boldsymbol{W}_V$ directly. Therefore, our Sliceformer has fewer parameters and lower computational complexity than the Transformer and its variants. We analyze the connections and differences between the proposed slicing-sorting operation and the MHA mechanism and discuss its rationality in depth. Specifically, we find that although abandoning the MHA architecture, the slicing-sorting operation achieves sparse and doubly stochastic attention maps by a set of permutation matrices. We test our Sliceformer in the well-known Long-Range Arena (LRA) benchmark. As shown in Figure 1, our Sliceformer achieves superior performance with less memory cost and runtime compared to the Transformer and its variants. Moreover, through other discriminative learning tasks like image classification and molecular analysis, we further demonstrate the universal applicability of our Sliceformer and highlight its advantage in numerical stability.

2 Related Work

Transformer [1] is a powerful sequential model friendly to parallel computing. Since it was proposed, Transformer has become the critical module of many large language models, e.g., BERT [2], Transformer-XL [5], and GPT [3]. Besides texts, Transformer is also applied to other sequences, e.g., the Music Transformer [26] for music modeling, the Informer [9] for time series broadcasting and the Transformer Hawkes process [27] for event sequence prediction. For non-sequential data like images, the Vision Transformer (ViT) [6] and its variants [7, 28] take the patches of images as a sequence and extract image representations accordingly, which outperform convolutional neural networks (CNNs) in image classification. Nowadays, Transformer is being introduced for structured data modeling, e.g., the Graphormer [10] for molecules, the Set-Transformer [11] for point clouds, and the Mesh Transformer [29] for 3D meshes. The more applications the Transformer has, the more significant it is to study its architecture, especially its MHA mechanism.

It has been known that the classic MHA mechanism suffers from high computational complexity, poor scalability, and numerical instability for long sequences. As shown in Table 1, many efforts have been made to overcome these issues. The SparseTrans in [4] and the Longformer in [13] compute local attention maps based on the sub-sequences extracted by sliding windows, which leads to sparse global attention maps. Some other models sparsify the key and query matrices directly by the locality-sensitive hashing (LSH) [14] or the ReLU operation [15]. Besides pursuing sparse attention maps, another technical route is constructing low-rank attention maps. The Performer [16] reduces the feature dimension (the column number) of the query and key matrices, while the Linformer [17] reduces the sample dimension (the row number) of the key and value matrices.

In addition to simplifying the computation of the attention maps, some work provides new understandings of the attention mechanism. The work in [19] treats the attention map as a normalized linear kernel and revisits the vanilla Transformer through different kernels. The Performer [16] and the CosFormer [15] introduce additional mappings for the query and key matrices and consider their linear kernels in the latent spaces. Recently, the work [18] reports an interesting phenomenon that the attention map tends to be doubly stochastic during training. Accordingly, it implements the attention map as an optimal transport through the Sinkhorn-Knopp algorithm [30]. Note that although providing these new understandings, these Transformer variants fail to design new model architectures that overcome the MHA's issues.

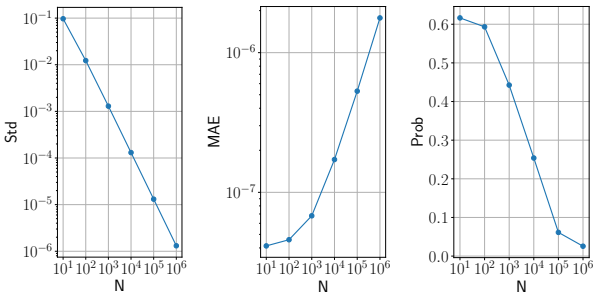
3 Proposed Sliceformer

3.1 Motivations

Typically, given an input $\boldsymbol{X} \in \mathbb{R}^{N\times d}$, where N indicates the length of a sequence or the size of a sample set and d is the input feature dimension, an attention head first obtains the value, query, and key matrices by linear maps, i.e., $\boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}_V \in \mathbb{R}^{N\times D}$, $\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}_Q \in \mathbb{R}^{N\times D}$, and

Algorithm 1 Numerical Test of Softmax

```
x = torch.randn(N)
index = torch.randperm(N)
y1 = torch.softmax(x)[index]
y2 = torch.softmax(x[index])
Std = torch.std(y1)
MAE = torch.norm(y1-y2, p=1)
Prob = torch.sum(y1==y2) / N
```



(a) (b) (c)

Figure 2: Algorithm 1 shows the PyTorch code used to verify the numerical issues of the softmax operation. Setting the sequence length $N \in \{10, ..., 10^6\}$, we run the code 100 trials with Pytorch 2.0.0 and Python 3.9. With the increase of N, (a) the softmax operation suffers from the over-smoothness issue, and (b, c) the exchange of the softmax operation and the permutation operation destructs the permutation-equivariance of output numerically.

$\boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}_K \in \mathbb{R}^{N\times D}$, and then projects $\boldsymbol{V}$ as follows:

$$\text{Attention}(\boldsymbol{V}; \boldsymbol{Q}, \boldsymbol{K}) := \boldsymbol{P}(\boldsymbol{Q}, \boldsymbol{K})\boldsymbol{V}. \tag{1}$$

Here, we take $\boldsymbol{V}$ as the input of the head, and $\boldsymbol{P}(\boldsymbol{Q}, \boldsymbol{K}) \in \mathbb{R}^{N\times N}$ is the attention map parametrized by $\boldsymbol{Q}$ and $\boldsymbol{K}$. The multi-head attention layer applies a group of linear maps, i.e., $\theta = \{\boldsymbol{W}_{V,m}, \boldsymbol{W}_{Q,m}, \boldsymbol{W}_{K,m} \in \mathbb{R}^{d\times D}\}_{m=1}^{M}$, to construct M attention heads and concatenates their outputs, i.e.,

$$\begin{aligned} &\text{MHA}_\theta(\boldsymbol{X}) := \text{Concat}_{\text{row}}(\{\text{Attention}(\boldsymbol{V}_m; \boldsymbol{Q}_m, \boldsymbol{K}_m)\}_{m=1}^{M}) \in \mathbb{R}^{N\times MD}, \\ &\text{where } \boldsymbol{V}_m = \boldsymbol{X}\boldsymbol{W}_{V,m},\ \boldsymbol{Q}_m = \boldsymbol{X}\boldsymbol{W}_{Q,m},\ \text{and } \boldsymbol{K}_m = \boldsymbol{X}\boldsymbol{W}_{K,m},\ \forall m = 1, ..., M. \end{aligned} \tag{2}$$

In this study, we aim to propose a surrogate of the MHA mechanism in (2). The design of our model is motivated by the following observations and analysis.

Numerical Issues of Softmax. Table 1 shows that most existing attention maps are implemented based on the softmax operation. The softmax operation makes the attention maps dense and row-stochastic, which suffers from numerical issues when dealing with long sequences. Firstly, it has been well-known that the output of the softmax operation tends to be over-smoothed with the increase of data size — the standard deviation of the output elements reduces rapidly, as shown in Figure 2(a). Without any additional data preprocessing like LSH [14] or sliding windows [13, 4], the softmax operation always leads to over-smoothed attention maps for long sequences.

Secondly, we observe a numerical issue seldom considered by existing work, called the "destruction of permutation-equivariance (DPE)". In particular, the softmax operation should be permutation-equivariant in theory, i.e., $\text{Softmax}_\sigma(\boldsymbol{x}) = \text{Softmax}(\boldsymbol{x}_\sigma)$ for an arbitrary vector $\boldsymbol{x}$ and its index permutation σ. To our surprise, however, this property cannot be held in practice — the exchange of the softmax operation and the permutation operation leads to different outputs, as shown in Figures 2(b) and 2(c). With the increase in data size, the mean absolute error between the outputs increases, and the probability of appearing identical elements decreases. Stacking multiple softmax-based attention layers may make the DPE issue more severe. Suppressing the DPE problem is necessary for the Transformers modeling structured data as sequences (e.g., the ViT [6] modeling images as sequences of patches) because the attention maps destructing the permutation-equivariance property may result in permutation-variant data representations.

Recent Advances in Attention Models. As Figure 1, the following experiments, and the results in [13, 25, 15, 22] show, the models applying sparse attention maps, e.g., CosFormer [15] and Longformer [13], can achieve competitive performance and higher efficiency than the vanilla Transformer. On the contrary, although imposing low-rank structures on the attention maps can also reduce the computational complexity, it seems to harm the performance — the gaps between the vanilla Transformer and the models using low-rank attention maps (e.g., Local Attention [31] and Linear Trans. [23]) are significant on the LRA benchmark. In our opinion, a potential reason for this phenomenon is that when $\text{rank}(\boldsymbol{P}) \ll N$, the rank of the output embeddings $\text{MHA}_\theta(\boldsymbol{X})$ is likely to be smaller than N.

The low-rank structure may limit the representation power of the output embeddings. In addition, the recent work in [18] shows that in various discriminative learning tasks, the attention maps tend to be doubly stochastic (i.e., $\boldsymbol{P}\mathbf{1}_N = \mathbf{1}_N$ and $\boldsymbol{P}^\top \mathbf{1}_N = \mathbf{1}_N$) during training.[1] Accordingly, the Sinkformer designs the attention maps based on the Sinkhorn-Knopp algorithm [30] and makes them doubly stochastic strictly, achieving better performance than the vanilla Transformer.

Restricted Capacity of Multi-Head Architecture. In our opinion, the multi-head architecture is designed to balance model capacity and computational complexity. The MHA projects the input to different latent spaces and applies one attention map to each value matrix [1], as shown in (2). The features within a value matrix, i.e., the columns of $\boldsymbol{V}$, share the same attention map $\boldsymbol{P}$, so we only need to compute one attention map for D features. Although such an architecture has better capacity than the single-head attention layer, it does not maximize the capacity of the attention layer. In particular, given MD features $\{\boldsymbol{V}_m \in \mathbb{R}^{N\times D}\}_{m=1}^{M}$, computing a specialized attention map for each feature can maximize the model capacity but leads to high computational complexity. Therefore, the solution of the multi-head attention layer is to group the features by M heads and let the features in the same head share the same attention map. In other words, the multi-head attention sacrifices some model capacity for computational efficiency.

3.2 Design Principle and Implementations

Based on the above observations and analysis, we propose the following two hypotheses to guide the design of our model.

Hypothesis 1. *A desired attention map $\boldsymbol{P}$ should be full-rank, sparse, and doubly stochastic.*

Hypothesis 2. *The multi-head architecture aims to achieve a trade-off between model capacity and computational complexity. It might be unnecessary if we can find a surrogate that simultaneously has high capacity and low complexity.*

Hypothesis 1 is about the desired structure of the attention map. It implies that i) the usage of the softmax operation should be avoided, and ii) the parameterization based on the query and key matrices might be unnecessary as long as the attention map can keep the desired structure and be updated during training. Hypothesis 2 is about the necessity of the multi-head architecture. It implies that if we can compute a specialized attention map for each feature with low computational cost, such an operation might be comparable, even superior, to the current MHA mechanism.

We design our Sliceformer based on these two hypotheses. As shown in Table 1, the key contribution of our Sliceformer is implementing a new attention layer based on an extremely simple **slice-sorting** operation, which projects the input linearly to a latent space and sorts each feature, i.e.,

$$\text{SliceSort}(\boldsymbol{X}) := \text{Sort}_{\text{col}}(\underbrace{\boldsymbol{X}\boldsymbol{W}_V}_{\boldsymbol{V}=[\boldsymbol{v}_i]}) = \text{Concat}_{\text{row}}(\{\boldsymbol{P}_i\boldsymbol{v}_i\}_{i=1}^{MD}) \in \mathbb{R}^{N\times MD}, \tag{3}$$

where $\boldsymbol{W}_V \in \mathbb{R}^{d\times MD}$ is the projection matrix,[2] and $\boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}_V$. Here, we call each column of $\boldsymbol{V}$ a "slice", denoted as $\boldsymbol{v}_i$ for $i = 1, ..., MD$. Each slice corresponds to the projection result of N d-dimensional samples in a 1D space. As shown in (3), sorting a slice $\boldsymbol{v}_i$ corresponds to the multiplication between a permutation matrix $\boldsymbol{P}_i$ and the slice, and accordingly, our slicing-sorting operation concatenates all sorted slices as the output.

Our slicing-sorting attention layer has some potential advantages compared to the MHA mechanism.

Structured and Efficient Attention Map. As shown in (3), our slicing-sorting operation implements a permutation matrix as an attention map for each feature dimension. The permutation matrix naturally has the full-rank, sparse, and doubly stochastic structure proposed in Hypothesis 1, leading to a competitive alternative to traditional attention maps. Moreover, instead of generating an explicit attention map with size $\mathcal{O}(N^2)$, we implement permutation matrix implicitly via sorting, whose space complexity is $\mathcal{O}(\log N)$ in general and $\mathcal{O}(N)$ in the worst case.

Low Complexity and High Capacity. Our slicing-sorting operation has high model capacity because it generates a specific attention map for each feature dimension rather than a feature group. Given

[1] See Figure 2 in [18] for more details.

[2] Here, we set the number of columns in $\boldsymbol{W}_V$ to be MD, such that the output has the same shape with the output of MHA.

$\boldsymbol{V} \in \mathbb{R}^{N\times MD}$, we sort the MD columns, whose computational complexity is $\mathcal{O}(MDN\log N)$. On the contrary, the other attention layers merely generate M attention maps, and their computational complexity (i.e., the complexity per head in Table 1 times the number of heads) is at most comparable to ours. According to Hypothesis 2, our slicing-sorting attention layer has the potential to outperform the multi-head architecture.

Permutation-Invariance. The formulations in Table 1 indicate that traditional attention layers are permutation-equivariant in theory. Given the outputs of such layers, we can obtain permutation-invariant data representations by $i)$ pooling the outputs and $ii)$ extracting the outputs of predefined super nodes [10] or "CLS" positions [1, 6]. As aforementioned, the permutation-equivariance cannot be held numerically for long sequences because of the DPE problem of the softmax operation. As a result, the permutation-invariance of the representation may also suffer from numerical errors. On the contrary, the output of our slicing-sorting attention layer is naturally permutation-invariant because $\text{Sort}_{\text{col}}(\boldsymbol{V}) = \text{Sort}_{\text{col}}(\boldsymbol{V}_\sigma)$ for an arbitrary row index permutation σ. The simplicity of our operation ensures that this permutation-invariance can be held without numerical errors.

3.3 Variants of Sliceformer

Replacing the MHA of the Transformer with our slicing-sorting attention layer leads to our Sliceformer. Note that, besides sorting, we can obtain attention matrices by many other methods. Here, we provide the following two methods, leading to two variants of Sliceformer.

1. **Multi-permutation:** Given the permutation matrix $\boldsymbol{P}$ derived by sorting, we can derive more permutation matrices by $\boldsymbol{P}^k$, $k \in \mathbb{Z}_+$. Accordingly, for each feature $\boldsymbol{v}_i$, we can treat $\sum_{k=1}^{K} \alpha_k \boldsymbol{P}_i^k$ as its attention map, where $\boldsymbol{\alpha} = [\alpha_k]$ is in the $(K-1)$-Simplex. We can learn $\boldsymbol{\alpha}$ or set it in advance. The attention map $\sum_{k=1}^{K} \alpha_k \boldsymbol{P}_i^k \boldsymbol{v}_i$ can be implemented implicitly by sorting $\boldsymbol{v}_i$ once, permuting its elements K times based on the same index order, and averaging the results. Its computational complexity is comparable to the sorting operation.
2. **Max-exchange:** For each feature $\boldsymbol{v}_i$, we find its maximum element and exchange it with the first element, whose computational complexity is $\mathcal{O}(N)$ in time and $\mathcal{O}(1)$ in space.

In the following experiments, the variants of our Sliceformer help to provide insights into the performance of our Sliceformer and the rationality of the proposed hypotheses.

4 Experiments

We demonstrate the effectiveness and efficiency of our Sliceformer through comprehensive comparative and analytic experiments. In particular, we first compare our Sliceformer to Transformer and its representative variants on the Long Range Arena benchmark [31] and empirically verify the rationality of our slicing-sorting operation. Then, we implement ViT [6] by our Sliceformer and test its performance and permutation-invariance in image classification tasks. Finally, we impose our slice-sorting operation on Graphormer [10], leading to a Sliceformer for graph-structured data. This model achieves comparable performance in molecular representation with fewer parameters, demonstrating the universal applicability of our slice-sorting attention layer. **Representative results are shown below. More results and implementation details are in the supplementary file.**

4.1 Long Range Arena Benchmark

Long Range Arena (LRA) is a benchmark designed to evaluate models for long sequence scenarios [31], which consists of six discriminative learning tasks, including ListOps [32], byte-level text classification [33], byte-level document retrieval [34], and three image classification tasks on CIFAR-10 [35], Pathfinder [36],[3] and Pathfinder-X (a longer and more challenging version of Pathfinder). In the three image classification tasks, each image is formulated as a sequence of pixels. We test our Sliceformer on the LRA benchmark and compare it to other Transformer-based models on both prediction accuracy and computational efficiency. In each task, our Sliceformer is trained to represent each input sequence through the embedding in the "CLS" position. For a fair comparison,

[3] Pathfinder is an image classification task: given a set of gray-level images, each of which plots two points and several curves, the model aims to recognize whether there exists a path connecting the points in each image.

Table 2: The comparison for various models on the LRA benchmark. "FAIL" means the training process fails to converge. In each column, we bold the best result and underline the second best one.

Model	ListOps	Text	Retrieval	Image	Path	Path-X	Avg. Acc (%)
Transformer [1]	36.37	64.27	57.46	42.44	71.40	FAIL	54.39
Local Attention [31]	15.82	52.98	53.39	41.46	66.63	FAIL	46.06
Linear Trans. [23]	16.13	**65.90**	53.09	42.34	75.30	FAIL	50.55
Reformer [14]	37.27	56.10	53.40	38.07	68.50	FAIL	50.67
Sinkformer [18]	30.70	64.03	55.45	41.08	64.65	FAIL	51.18
Sparse Trans. [4]	17.07	63.58	59.59	<u>44.24</u>	71.71	FAIL	51.24
Sinkhorn Trans. [20]	33.67	61.20	53.83	41.23	67.45	FAIL	51.29
Linformer [17]	35.70	53.94	52.27	38.56	76.34	FAIL	51.36
Performer [16]	18.01	<u>65.40</u>	53.82	42.77	<u>77.05</u>	FAIL	51.41
Synthesizer [38]	36.99	61.68	54.67	41.61	69.45	FAIL	52.88
Longformer [13]	35.63	62.85	56.89	42.22	69.71	FAIL	53.46
BigBird [39]	36.05	64.02	59.29	40.83	74.87	FAIL	55.01
Cosformer [15]	**37.90**	63.41	<u>61.36</u>	43.17	70.33	FAIL	<u>55.23</u>
Sliceformer	<u>37.65</u>	64.60	**62.23**	**48.02**	**82.04**	FAIL	**58.91**

Table 3: The comparison for various models on their computational efficiency. "OOM" means the training process suffers from the out-of-memory issue. In each column, we bold the best result and underline the second best one.

Model	Training speed (steps per second)				Peak Memory Usage (GB)			
	1K	2K	3K	4K	1K	2K	3K	4K
Transformer [1]	27.49	9.45	4.73	OOM	11.64	32.45	65.67	OOM
Local Attention [31]	31.41	25.47	18.06	13.80	6.23	9.24	12.26	15.27
Linear Trans. [23]	31.35	25.79	17.07	12.32	6.50	9.84	13.18	16.52
Reformer [14]	31.55	22.00	13.42	8.84	6.91	12.22	19.54	28.88
Sinkformer [18]	18.72	5.82	2.86	OOM	14.13	41.58	82.53	OOM
Sparse Trans. [4]	27.39	9.47	4.72	OOM	11.64	32.45	65.71	OOM
Sinkhorn Trans. [20]	29.88	22.00	15.66	11.70	6.70	10.23	13.78	17.32
Linformer [17]	28.47	<u>28.08</u>	<u>19.08</u>	<u>14.55</u>	<u>5.95</u>	<u>8.82</u>	<u>11.65</u>	<u>14.47</u>
Performer [16]	28.84	26.67	18.97	14.10	6.25	9.35	12.45	15.54
Synthesizer [38]	19.57	6.27	3.01	OOM	12.77	37.53	75.23	OOM
Longformer [13]	17.56	5.67	2.74	OOM	13.00	37.07	75.46	OOM
BigBird [39]	27.27	14.34	9.76	7.21	9.59	16.53	23.16	30.07
Cosformer [15]	<u>32.42</u>	26.13	17.56	12.58	6.36	9.68	13.36	16.95
Sliceformer	**32.79**	**32.26**	**21.79**	**16.25**	**5.61**	**8.12**	**10.64**	**13.14**

we implement all the models based on JAX [37] and strictly follow the benchmark's default data processing and experimental design. In each task, our Sliceformer is the same as the other models on the number of layers and the dimension of hidden variables. Hence, its model parameters are fewer because of abandoning the "query-key-value" attention architecture. All the models are trained on 4 NVIDIA 3090 GPUs.

As shown in Table 2, Sliceformer performs the best in three of the six tasks and achieves the highest average accuracy. Especially in the challenging image classification tasks on CIFAR-10 and Pathfinder, our Sliceformer outperforms the other models significantly, improving the classification accuracy by at least four percentage points. Table 3 compares the models' training speed and peak memory usage when dealing with the sequences with lengths ranging from 1K to 4K. The most efficient model among the baselines is Linformer [17], but its average accuracy on LRA is merely 51.36%. Our Sliceformer is more efficient than the other Transformer-based models, which can execute more training steps per second and occupy less memory. The advantage of our Sliceformer on computational efficiency becomes even more significant with the increase of the sequence length. In summary, when modeling long sequences, our Sliceformer can achieve comparable or superior performance with significant improvements in computational efficiency compared to the Transformer and its variants. The experimental results have also been illustrated in Figure 1.

Table 4: The comparison for various sorting mechanisms on the Long Range Arena benchmark.

Sorting Order	ListOps	Text	Retrieval	Image	Path	Path-X	Avg. Acc (%)
Ascending (Default)	37.65	64.60	62.23	48.02	82.04	FAIL	58.91
Descending	37.50	64.36	61.95	45.46	81.70	FAIL	58.19
Half-Half	37.45	64.25	61.95	45.80	81.38	FAIL	58.17
Multi-permutation ($K = 2$)	37.25	63.70	58.42	47.37	81.04	FAIL	57.56
Multi-permutation ($K = 4$)	37.35	63.80	57.16	46.92	74.06	FAIL	55.86
Max-exchange	37.00	62.90	59.00	40.48	75.42	FAIL	54.96
Shuffle	17.85	50.49	50.87	10.00	49.74	FAIL	35.79

Essentially, the slicing-sorting operation in our Sliceformer derives as many attention maps as possible based on sorting. Each attention map is determined by the corresponding data column of the value matrix and the sorting order we set. Given a value matrix $\boldsymbol{V} \in \mathbb{R}^{N \times MD}$, our Sliceformer sorts its columns in ascending order by default. To quantitatively analyze the impacts of different sorting mechanisms on Sliceformer, we further consider five different settings: i) sorting the columns of $\boldsymbol{V}$ in descending order; ii) sorting $\frac{MD}{2}$ columns of $\boldsymbol{V}$ in ascending order and sorting the remaining columns in descending order (denoted as "Half-Half"); iii) the "multi-permutation" method mentioned in Section 3.3; iv) the "max-exchange" method mentioned in Section 3.3; and v) shuffling the rows of $\boldsymbol{V}$ (i.e., permuting all the columns in the same order).

Table 4 compares the performance of our Sliceformer achieved under different sorting mechanisms. The experimental results empirically demonstrate the rationality of our slicing-sorting operation and verify the correctness of the hypotheses that support our design principle. Firstly, **our Sliceformer is robust to the sorting order.** The performance achieved under ascending, descending, and half-half settings is comparable. A potential reason is that these three settings can generate attention maps (i.e., permutation matrices) with sufficient diversity for the columns of $\boldsymbol{V}$. Secondly, **sparse attention maps work well for long sequence modeling.** When applying the multi-permutation strategy to generate attention maps, the performance of Sliceformer degrades. Moreover, the performance degradation becomes more severe with the increase of K. This phenomenon may be explained based on Hypothesis 1: although the attention maps obtained by the multi-permutation strategy are full-rank and doubly stochastic, each of them contains $\mathcal{O}(KN)$ nonzero elements, which is not so sparse as a single permutation matrix. Thirdly, **the diversity of attention maps matters, which impacts the model capacity significantly.** When applying the max-exchange strategy leads to performance degradation as well. Compared to the sorting operation, this strategy has lower complexity and can also generate attention maps that meet the requirement in Hypothesis 1. However, the generated attention maps have limited diversity because each is a permutation matrix merely exchanging a pair of elements. Applying these attention maps to the columns of $\boldsymbol{V}$, most rows of $\boldsymbol{V}$ are likely to be unchanged, especially for long sequences. Because of the same reason, the shuffle strategy leads to the catastrophic degradation of learning results — this strategy applies a single attention map to all columns of $\boldsymbol{V}$. As a result, both the max-exchange and the shuffle strategies restrict the model capacity and thus disobey Hypothesis 2.

Numerical Permutation-Invariance. For the CIFAR-10 image classification task in LRA, we further compare our Sliceformer with the vanilla Transformer on their numerical permutation-invariance. Given a pixel sequence of an image and its randomly-shuffled version, we pass them through each model and compute the MAE between the corresponding outputs. The results in Figure 3(a) illustrate the advantage of our Sliceformer. With the increase of depth (i.e., the number of stacked attention layers), the output of the Transformer is not permutation-invariant numerically. On the contrary, our Sliceformer can preserve the permutation-invariance property perfectly. This phenomenon is because the stacking of the softmax operations leads to the propagation of numerical errors in the Transformer. Our Sliceformer avoids this problem by abandoning the softmax.

4.2 More Applications

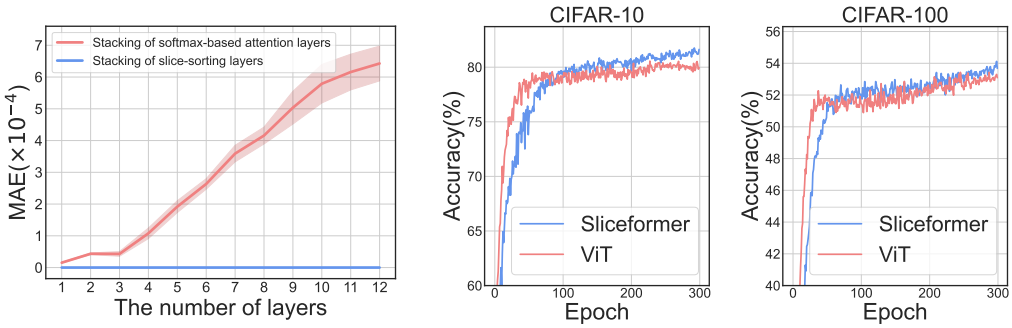
Image Classification. Like ViT [6], we can treat images as patch sequences and apply our Sliceformer to achieve image classification. We test our Sliceformer on three image datasets, including CIFAR-10, CIFAR-100 [35], Tiny-ImageNet [40]. We compare our Sliceformer to ViT for each dataset on their model size and classification accuracy. For each dataset, we train the models from scratch. Table 5

Table 5: The comparison for our Sliceformer and ViT on the number of parameters ($\times 10^6$) and classification accuracy (%).

Data	CIFAR-10		CIFAR-100		Tiny-ImageNet	
	Size	Top-1 Acc	Size	Top-1 Acc	Size	Top-1 Acc
ViT	9.60	80.98	9.65	53.99	22.05	**52.74**
Sliceformer	**6.46**	**82.16**	**6.50**	**54.24**	**18.50**	51.77

Table 6: The comparison on the number of parameters ($\times 10^6$) and prediction error (MAE).

Model	Size	MAE
Graphormer	47.09	**0.1287**
Sliceformer	**32.91**	0.1308



(a) Permutation-invariance test

(b) Convergence of training process

Figure 3: (a) The impact of the order of patch sequence on the permutation-invariance of image representation. (b) The comparison for our Sliceformer and ViT on their convergence when training on the CIFAR-10 and CIFAR-100 dataset.

compares the classification accuracy achieved by the two models. Our Sliceformer outperforms ViT consistently on both model size and classification accuracy for CIFAR-10 and CIFAR-100. Additionally, Figure 3(b) shows the training convergence of the two models. For Tiny-ImageNet, our Sliceformer achieves comparable classification accuracy with fewer parameters.

Molecular Analysis. Besides achieving a vision Sliceformer, we introduce the slicing-sorting operation to Graphormer [10] and test its impacts on molecular property prediction. In particular, the attention head of Graphormer applies a modified "query-key-value" architecture, which is formulated as $\text{Softmax}(\boldsymbol{S} + \boldsymbol{E} + \frac{1}{\sqrt{D}}\boldsymbol{Q}\boldsymbol{K}^\top)\boldsymbol{V}$. Here $\boldsymbol{S}$ and $\boldsymbol{E}$ are learnable embedding matrices encoding spatial positions and edge information, respectively. Applying the slicing-sorting operation, we design a simplified attention layer without the query and key matrices and dramatically reduce the trainable parameters by around 30%. Replacing the attention layer of Graphormer with this layer leads to a Sliceformer for molecular data. We apply the PCQM4M-LSC dataset [41] for training and testing the models. The experimental results in Table 6 show that applying the slicing-sort operation leads to a simplified model with a smaller size, whose performance is comparable to Graphormer.

5 Conclusion and Future Work

We have proposed a new data representation model called Sliceformer. By replacing the traditional MHA mechanism with a simple slicing-sorting operation, our Sliceformer overcomes the numerical drawbacks of the current Transformers and achieves encouraging performance in various discriminative tasks. **Our work provides a new perspective to the design of the attention layer, i.e., implementing attention maps through simple algorithmic operations has the potential to achieve high model capacity, low computational complexity, and good numerical performance.**

Limitations and Future Work. Currently, the rationality of the proposed model is based on the proposed hypotheses and empirical experiments, and its performance in generative learning tasks is not investigated yet. Therefore, we would like to find theoretical support for our model and test it in generative learning tasks in the future. Additionally, beyond the slicing-sorting operation, we plan to develop other better-performing attention layers and apply them to the models for structured data like point clouds and meshes.

References

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, 2019.

[3] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[4] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

[5] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G Carbonell, Quoc Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2978–2988, 2019.

[6] 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. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=YicbFdNTTy`.

[7] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 10012–10022, 2021.

[8] Anurag Arnab, Mostafa Dehghani, Georg Heigold, Chen Sun, Mario Lučić, and Cordelia Schmid. Vivit: A video vision transformer. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 6836–6846, 2021.

[9] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(12):11106–11115, 2021.

[10] Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform badly for graph representation? *Advances in Neural Information Processing Systems*, 34:28877–28888, 2021.

[11] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International conference on machine learning*, pages 3744–3753. PMLR, 2019.

[12] Hengshuang Zhao, Li Jiang, Jiaya Jia, Philip HS Torr, and Vladlen Koltun. Point transformer. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 16259–16268, 2021.

[13] Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

[14] Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *International Conference on Learning Representations*, 2020.

[15] Zhen Qin, Weixuan Sun, Hui Deng, Dongxu Li, Yunshen Wei, Baohong Lv, Junjie Yan, Lingpeng Kong, and Yiran Zhong. cosformer: Rethinking softmax in attention. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=Bl8CQrx2Up4`.

[16] Krzysztof Marcin Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Quincy Davis, Afroz Mohiuddin, Lukasz Kaiser, David Benjamin Belanger, Lucy J Colwell, and Adrian Weller. Rethinking attention with performers. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=Ua6zuk0WRH`.

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

[18] Michael E Sander, Pierre Ablin, Mathieu Blondel, and Gabriel Peyré. Sinkformers: Transformers with doubly stochastic attention. In *International Conference on Artificial Intelligence and Statistics*, pages 3515–3530. PMLR, 2022.

[19] Yao-Hung Hubert Tsai, Shaojie Bai, Makoto Yamada, Louis-Philippe Morency, and Ruslan Salakhutdinov. Transformer dissection: An unified understanding for transformer's attention via the lens of kernel. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4344–4353, 2019.

[20] Yi Tay, Dara Bahri, Liu Yang, Donald Metzler, and Da-Cheng Juan. Sparse sinkhorn attention. In *International Conference on Machine Learning*, pages 9438–9447. PMLR, 2020.

[21] Yihe Dong, Jean-Baptiste Cordonnier, and Andreas Loukas. Attention is not all you need: Pure attention loses rank doubly exponentially with depth. In *International Conference on Machine Learning*, pages 2793–2803. PMLR, 2021.

[22] Xuezhe Ma, Chunting Zhou, Xiang Kong, Junxian He, Liangke Gui, Graham Neubig, Jonathan May, and Luke Zettlemoyer. Mega: moving average equipped gated attention. *arXiv preprint arXiv:2209.10655*, 2022.

[23] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International Conference on Machine Learning*, pages 5156–5165. PMLR, 2020.

[24] Ilya O Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, et al. Mlp-mixer: An all-mlp architecture for vision. *Advances in neural information processing systems*, 34: 24261–24272, 2021.

[25] Albert Gu, Karan Goel, and Christopher Re. Efficiently modeling long sequences with structured state spaces. In *International Conference on Learning Representations*, 2022.

[26] Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Ian Simon, Curtis Hawthorne, Noam Shazeer, Andrew M. Dai, Matthew D. Hoffman, Monica Dinculescu, and Douglas Eck. Music transformer. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=rJe4ShAcF7`.

[27] Simiao Zuo, Haoming Jiang, Zichong Li, Tuo Zhao, and Hongyuan Zha. Transformer hawkes process. In *International conference on machine learning*, pages 11692–11702. PMLR, 2020.

[28] Zhengsu Chen, Lingxi Xie, Jianwei Niu, Xuefeng Liu, Longhui Wei, and Qi Tian. Visformer: The vision-friendly transformer. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 589–598, 2021.

[29] Kevin Lin, Lijuan Wang, and Zicheng Liu. Mesh graphormer. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 12939–12948, 2021.

[30] Richard Sinkhorn and Paul Knopp. Concerning nonnegative matrices and doubly stochastic matrices. *Pacific Journal of Mathematics*, 21(2):343–348, 1967.

[31] Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. Long range arena: A benchmark for efficient transformers. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=qVyeW-grC2k`.

[32] Nikita Nangia and Samuel R Bowman. Listops: A diagnostic dataset for latent tree learning. *arXiv preprint arXiv:1804.06028*, 2018.

[33] Andrew Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*, pages 142–150, 2011.

[34] Dragomir R Radev, Pradeep Muthukrishnan, Vahed Qazvinian, and Amjad Abu-Jbara. The acl anthology network corpus. *Language Resources and Evaluation*, 47:919–944, 2013.

[35] Alex Krizhevsky. Learning multiple layers of features from tiny images. *Master's thesis, University of Toronto*, 2009.

[36] Drew Linsley, Junkyung Kim, Vijay Veerabadran, Charles Windolf, and Thomas Serre. Learning long-range spatial dependencies with horizontal gated recurrent units. *Advances in neural information processing systems*, 31, 2018.

[37] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, et al. Jax: composable transformations of python+ numpy programs. 2018.

[38] Yi Tay, Dara Bahri, Donald Metzler, Da-Cheng Juan, Zhe Zhao, and Che Zheng. Synthesizer: Rethinking self-attention for transformer models. In *International conference on machine learning*, pages 10183–10192. PMLR, 2021.

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

[40] Ya Le and Xuan Yang. Tiny imagenet visual recognition challenge. *CS 231N*, 7(7):3, 2015.

[41] Weihua Hu, Matthias Fey, Hongyu Ren, Maho Nakata, Yuxiao Dong, and Jure Leskovec. Ogb-lsc: A large-scale challenge for machine learning on graphs. *arXiv preprint arXiv:2103.09430*, 2021.