PermuteFormer: Efficient Relative Position Encoding for Long Sequences

Peng Chen
Peking University
chen.peng@pku.edu.cn

Abstract

A recent variation of Transformer, Performer, scales Transformer to longer sequences with a linear attention mechanism. However, it is not compatible with relative position encoding, which has advantages over absolute position encoding. In this paper, we discuss possible ways to add relative position encoding to Performer. Based on the analysis, we propose PermuteFormer, a Performer-based model with relative position encoding that scales linearly on long sequences. PermuteFormer applies position-dependent transformation on queries and keys to encode positional information into the attention module. This transformation is carefully crafted so that the final output of self-attention is not affected by absolute positions of tokens. PermuteFormer introduces negligible computational overhead by design that it runs as fast as Performer. We evaluate PermuteFormer on Long-Range Arena, a dataset for long sequences, as well as WikiText-103, a language modeling dataset. The experiments show that PermuteFormer uniformly improves the performance of Performer with almost no computational overhead and outperforms vanilla Transformer on most of the tasks. [1]

1 Introduction

The Transformer architecture (Vaswani et al., 2017) has achieved state-of-the-art on various fields of research, including natural language processing (Devlin et al., 2019; Raffel et al., 2020), speech processing (Baevski et al., 2020) and image processing (Dosovitskiy et al., 2020; Tan and Bansal, 2019). But Transformer does not scale well to long sequences, because the time complexity and memory complexity of the attention module in Transformer are both quadratic to the sequence length. Recently, several efficient Transformers (Kitaev et al., 2020; Wang et al., 2020; Zaheer et al., 2020; Xiong et al., 2021) have been proposed to speed up the model from quadratic complexity to linear complexity without significant performance loss. Generally, they utilize efficient algorithms to approximate attention. §2 briefly introduces these efficient Transformers and a more thorough review can be found in Tay et al. (2020c).

Among these efficient Transformers, it is suggested that Performer (Choromanski et al., 2020) is the fastest one (Tay et al., 2020b). In this paper, we denote as Performer the family of efficient Transformers similar to Choromanski et al. (2020), e.g., Katharopoulos et al. (2020); Peng et al. (2021); Kasai et al. (2021); Likhosherstov et al. (2020), not only Choromanski et al. (2020) itself. Performer utilizes kernel method to avoid explicit calculation of attention weights. It applies a non-linear feature map to queries and keys to get query features and key features respectively and then multiplies query features, key features, and values together directly, without applying softmax. With the appropriate ordering of matrix multiplications, Performer achieves complexity linear of the sequence length. Moreover, some implementation of unidirectional Performer (Likhosherstov et al., 2020) even reduces memory footprint to constant at both training time and inference time.

Although Performer accelerates attention to linear complexity, the existing relative position encoding (Shaw et al., 2018; Dai et al., 2019; Raffel et al., 2020) still has quadratic complexity with respect to the sequence length. So Performer cannot benefit from relative position encoding, which has already been a common practice for a bunch of state-of-the-art Transformers (Yang et al., 2019; Raffel et al., 2020; He et al., 2020). Relative position encoding has several advantages over absolute position encoding. (1) Relative position encoding may be applied to sequences with arbitrary lengths, with no limitation imposed by training datasets. (2)

[1] Code is available at https://github.com/cpcp1998/PermuteFormer.

Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, pages 10606–10618
November 7–11, 2021. ©2021 Association for Computational Linguistics

Relative position encoding is more efficient and effective than absolute position encoding. (Shaw et al., 2018)

Besides Performer, existing relative position encodings also do not fit with other efficient Transformers. Some relative position encoding (Raffel et al., 2020) adds a bias to the attention matrix, and others (Shaw et al., 2018; Dai et al., 2019) add a relative-position-dependent bias to key vectors. Both require explicit calculation of dot-products between query vectors and key vectors. This conflicts with the second and third categories of efficient Transformers described in Section 2 because they reduce the computation complexity by avoiding the explicit calculation of dot-products between query vectors and key vectors. As for the first category of efficient Transformers, LSH in Kitaev et al. (2020) may fail to locate major attention weights in the presence of relative position encoding; Zaheer et al. (2020); Beltagy et al. (2020) rely on global tokens heavily, whose relative positions to other tokens are not defined.

In this paper, we propose a Performer-compatible relative position encoding that scales linearly on long sequences. Performer with this novel relative position encoding is named PermuteFormer. PermuteFormer applies a position-aware transformation on query features and key features to encode positional information. More specifically, we choose a random permutation $\pi : \{1, 2, \cdots, d\} \to \{1, 2, \cdots, d\}$ where d is the dimension of query / key features per attention head, and applies the permutation i times to i-th token's query / key feature.[2] In this way, positional information is encoded into attention weights. We prove that, although the transformation applied to query feature and key feature of a token depends on its absolute position, the effects of absolute position on query features and key features cancel out with each other on calculating dot-product of them. Thus, the final attention weights do not depend on the absolute positions, and PermuteFormer encodes relative position only.

PermuteFormer is as efficient as Performer, with negligible computational overhead. Permuting of query features and key features can be implemented efficiently, with computational complexity proportional to their size. Since the size is far less than the computational complexity of the whole model, the cost of permutation in PermuteFormer is negligible compared to the overall computational cost of Performer. The analysis above is also confirmed by the experiment results.

We evaluate PermuteFormer on Long-Range Arena (Tay et al., 2020b) for bidirectional case and on WikiText-103 (Merity et al., 2017) for unidirectional case. Long-Range Arena is a benchmark designed to evaluate efficient Transformers on long sequences. We find that the new relative position encoding improves the performance of PermuteFormer significantly on Long-Range Arena. It not only performs better than Performer but also out-performs the vanilla Transformer, as well as other efficient Transformers, e.g., Kitaev et al. (2020); Wang et al. (2020); Xiong et al. (2021). WikiText-103 is a language modeling dataset. PermuteFormer reduces the performance gap between Performer and Transformer on WikiText-103. It also speeds up the convergence of the model.

Contributions The main contribution of this paper is summarized as follows.

- We discuss possible ways to add relative position encoding to Performer. We theoretically propose three properties that Performer-compatible relative position encoding should hold.
- We introduce PermuteFormer, a Performer model with relative position encoding that scales linearly to long sequences. It permutes elements of query features and key features to encode positional information. It is the only Performer-compatible relative position encoding with linear complexity, as far as we know. PermuteFormer is as efficient as Performer.
- We conduct extensive experiments to evaluate PermuteFormer. It achieves strong empirical performance and obtains state-of-the-art on Long-Range Arena, a benchmark for efficient Transformers. It also improves the performance of Performer on language modeling tasks like WikiText-103.

2 Related Work

Efficient Transformers Transformers suffer from complexity quadratic to the sequence length. Various methods have been proposed to improve the efficiency of Transformers. We classify them

[2] When we say applying a permutation π to a vector $\mathbf{x} = [x_1, x_2, \cdots, x_d]$, we mean the operation maps $\mathbf{x}$ to vector $[x_{\pi(1)}, x_{\pi(2)}, \cdots, x_{\pi(d)}]$.

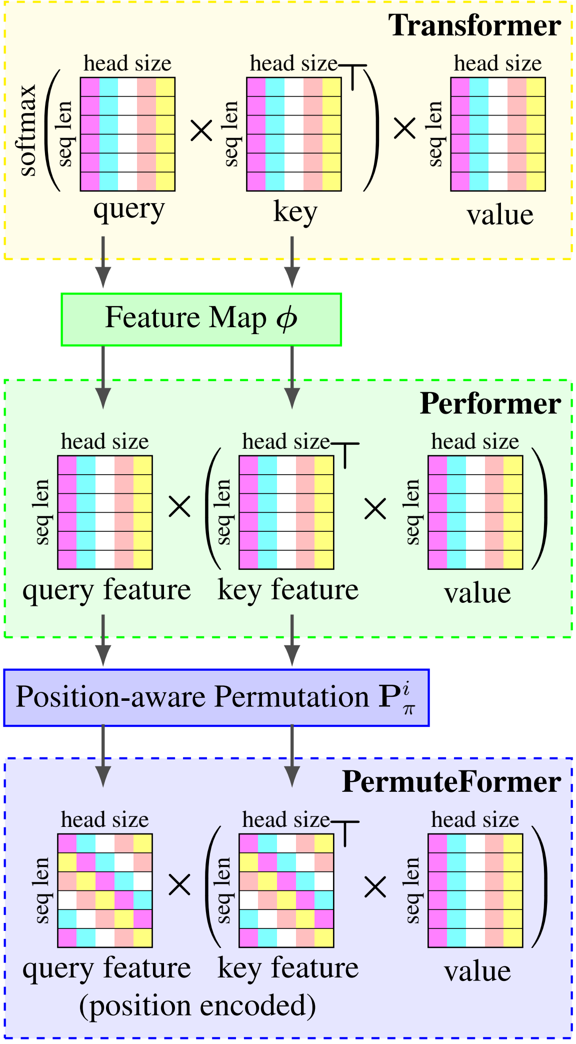

Figure 1: Attention in Transformer, Performer and PermuteFormer. Although attention is multi-headed in all of them, only one head is illustrated for clarity. **Transformer** applies softmax on dot-products of queries and keys to get the attention matrix, and then multiplies attention matrix and values to obtain outputs of attention module. **Performer** applies feature map, a non-linear projection, to queries and keys to get query features and key features. Then, it multiplies query features, key features and values from right to left. **PermuteFormer** applies a position-aware permutation on query features and key features first, and then do multiplications the same way as Performer. Each token's query / key feature is illustrated as a row of blocks in the figure, and its elements are marked with different colors. The position-aware permutation permutes elements of each token's query / key feature along the *head size* dimension in each attention head. Depending on the token's position, the permutation applied to query / key feature is different. Note that for Performer and PermuteFormer, only the numerator in Equation 11 is illustrated, as the denominator is simpler than the numerator.

into three categories. The first category of efficient Transformer omits the calculation of part of the attention matrix, exploiting the sparsity of the attention matrix. Kitaev et al. (2020) groups queries into buckets by local sensitive hash and computes intra-bucket attention weights only. Zaheer et al. (2020); Beltagy et al. (2020) limit attention matrix to specific sparse shapes. The second kind of efficient Transformers lowers matrix rank to reduce computation. Wang et al. (2020) projects keys and values to constant length independent of sequence lengths. Tay et al. (2020a) generates attention weights without keys. The third category of efficient Transformers, named Performer in this paper, leverages kernel methods to speed models up. Choromanski et al. (2020); Peng et al. (2021) view attention weights as kernel function of queries and keys, so they can be approximated by random features. Katharopoulos et al. (2020) relaxes the approximation requirement and finds that the model still works. Likhosherstov et al. (2020); Kasai et al. (2021) implement the unidirectional Performer as RNN so that their memory footprint is constant.

Relative Position Encoding Transformer itself does not capture the positional information of tokens, as it is invariant to permutations of tokens. Vaswani et al. (2017) solves this problem by adding a position embedding vector to the input of Transformer. Because the added position embedding depends on the absolute positions of tokens in a sequence, it is called absolute position encoding. For better representation of positional relation between tokens, Shaw et al. (2018) introduces relative position encoding to encode distances between tokens directly. There are two styles of relative position encoding. Shaw et al. (2018) adds relative position embedding to keys and values, while Dai et al. (2019) adds relative position embedding to queries and keys. Raffel et al. (2020), as the other style of relative position encoding, adds bias directly to the attention weights.

Concurrent Work Su et al. (2021) introduces RoFormer with a new kind of relative position encoding named RoPE, which is interoperable with Performer. Briefly, RoPE is a multiplicative sinusoidal absolute position embedding that rotates query (feature) vectors and key (feature) vectors according to their positions.

However, to make RoPE independent of absolute position, they sacrifice the property of attention matrices that every row sums to one. Moreover, they only discuss the possibility of integrating RoPE with Performer, but no experiment result is reported on such a model.

On the other hand, PermuteFormer's position encoding preserves the property of attention matrices mentioned above. In this paper, we compare the performance of PermuteFormer with RoFormer through experiment. The result shows that PermuteFormer fits the data better than RoFormer.

3 Methods

We propose an efficient relative position encoding that is compatible with Performer architecture. Performer with this new relative position encoding is named as PermuteFormer, because it permutes elements of query feature and key feature to encode positional information. The difference among vanilla Transformer, Performer and PermuteFormer is illustrated in Figure 1.

In this section, we first introduce Transformer and Performer briefly, and then describe details of PermuteFormer. For brevity and clarity, discussions in this section focus on a single head in multi-head attention. They can be directly applied to the whole multi-head attention.

3.1 Transformer and Performer

We give a brief introduction of Transformer and Performer's attention module in this section. Other parts of Transformer architecture (Vaswani et al., 2017) are omitted as they are unmodified in Performer and PermuteFormer.

The attention module in Transformer is a mapping from a sequence of vectors $\{\mathbf{x}_i^{\text{in}}\}_{i=1}^L$ to another sequence of vectors $\{\mathbf{x}_i^{\text{out}}\}_{i=1}^L$ with the same length L. In the attention module, the input vectors are first linearly mapped to three representations, named **query**, **key** and **value**. Formally,

$$\mathbf{q}_i = \mathbf{W}_q \mathbf{x}_i^{\text{in}},\ \mathbf{k}_i = \mathbf{W}_k \mathbf{x}_i^{\text{in}},\ \mathbf{v}_i = \mathbf{W}_v \mathbf{x}_i^{\text{in}}, \quad (1)$$

where $\mathbf{W}_q$, $\mathbf{W}_k$, $\mathbf{W}_v$ are transformation matrices for query, key and value, respectively. Then, similarities between queries and keys are calculated. The similarities are normalized to produce attention weights

$$\alpha_{ij} = \frac{\text{sim}(\mathbf{q}_i, \mathbf{k}_j)}{\sum_{l=1}^L \text{sim}(\mathbf{q}_i, \mathbf{k}_l)}, \quad (2)$$

where $\text{sim}(\mathbf{q}_i, \mathbf{k}_j)$ is the similarity of vector $\mathbf{q}_i$ and vector $\mathbf{k}_j$. Finally, output vectors $\mathbf{x}_i^{\text{out}}$ are obtained by weighted sum of values with weight $\{\alpha_{ij}\}_{i,j=1}^L$.

$$\mathbf{x}_i^{\text{out}} = \sum_{j=1}^L \alpha_{ij} \mathbf{v}_j. \quad (3)$$

Vanilla Transformer (Vaswani et al., 2017) adopts the following function as the similarity metric of queries and keys.

$$\text{sim}_{\text{Trans}}(\mathbf{q}_i, \mathbf{k}_j) = \exp\left(\mathbf{q}_i^\top \mathbf{k}_j / \sqrt{d}\right). \quad (4)$$

To reduce computation and memory cost, Performer's similarity function is approximated with kernel trick.

$$\text{sim}_{\text{Perf}}(\mathbf{q}_i, \mathbf{k}_j) = \boldsymbol{\phi}(\mathbf{q}_i)^\top \boldsymbol{\phi}(\mathbf{k}_j), \quad (5)$$

where $\boldsymbol{\phi}(\cdot)$ is a non-linear **feature map** from $\mathbb{R}^d$ to $\mathbb{R}^m$ for some model-specific m, so that the attention module can be expressed as follows.

$$\mathbf{x}_i^{\text{out}} = \left(\frac{\boldsymbol{\phi}(\mathbf{q}_i)^\top \sum_{j=1}^L \boldsymbol{\phi}(\mathbf{k}_j)\mathbf{v}_j^\top}{\boldsymbol{\phi}(\mathbf{q}_i)^\top \sum_{j=1}^L \boldsymbol{\phi}(\mathbf{k}_j)} \right)^\top. \quad (6)$$

We call $\boldsymbol{\phi}(\mathbf{q}_i)$ as **query feature** and $\boldsymbol{\phi}(\mathbf{k}_i)$ as **key feature**.

In this way, the $O(L^2)$ attention weight matrix is not explicitly calculated, so that the attention module costs only $O(L)$ time and memory, rather than the $O(L^2)$ complexity as vanilla Transformer. Different Performers differ by the choice of the mapping $\boldsymbol{\phi}(\cdot)$. A simple working choice is the ReLU function $\boldsymbol{\phi}(\mathbf{x}) = \max(\mathbf{x}, \mathbf{0})$ (Choromanski et al., 2020).

3.2 Relative Position Encoding for Performer

In this section, we discuss adding relative position encoding to Performer. We choose to modify the similarity function (Equation 5) to encode positional information. Specifically, we introduce an additional layer of position-dependent linear transformation over query features and key features. Now, the similary function becomes

$$\text{sim}_{\text{Perm}}(\mathbf{q}_i, \mathbf{k}_j) = \left(\mathbf{M}_i \boldsymbol{\phi}(\mathbf{q}_i)\right)^\top \left(\mathbf{N}_j \boldsymbol{\phi}(\mathbf{k}_j)\right), \quad (7)$$

where $\mathbf{M}_i, \mathbf{N}_j \in \mathbb{R}^{m \times m}$ are matrices parameterized by token's position i and j.

To ensure the similarity function depends only on the relative positions rather than absolute ones, $\mathbf{M}_i, \mathbf{N}_j$ must hold the following property.

Property 1 (Relative)**.** $\mathbf{M}_i^\top \mathbf{N}_j$ *is a function of* $i-j$*, i.e., it only depends on* $i-j$*.*

To prevent the similarity function from exploding as the sequence length grows, we have

Property 2 (Bounded)**.** *For a bidirectional model, there is an* B *that for all* $i, j \in \mathbb{Z}$*,* $\|\mathbf{M}_i^\top \mathbf{N}_j\| < B$*. For a unidirectional model, there is an* B *that for all* $i > j \in \mathbb{Z}$*,* $\|\mathbf{M}_i^\top \mathbf{N}_j\| < B$*.*

Additionally, the similarity function should be positive; otherwise, the model would be numerically unstable. If the similarity function alters between positive and negative values, in some cases, the denominator in Equation 2 may be zero while its numerator is not zero, leading the output of attention module tend to infinity. To keep the similarity function positive, one simple but efficient solution is to make all elements of query features and key features positive (Choromanski et al., 2020; Katharopoulos et al., 2020).

Property 3 (Positive)**.** *The linear transformations corresponding to matrix* $\mathbf{M}_i$ *and* $\mathbf{N}_j$ *map* $\mathbb{R}_+^m$ *to* $\mathbb{R}_+^m$*.*

We prove that, $\mathbf{M}_i^\top \mathbf{N}_j$ must be in a specific form to fulfill the requirement of Property 1.

Proposition 1. *Let* $\{\mathbf{M}_i\}_{i=-\infty}^{\infty}$ *be a series of* $l \times m$ *matrices,* $\{\mathbf{N}_i\}_{i=-\infty}^{\infty}$ *be a series of* $l \times n$ *matrices. Then,* $\mathbf{M}_i^\top \mathbf{N}_j$ *only depends on* $i-j$*, if and only if that, there is an integer* l'*, matrices* $\mathbf{R} \in \mathbb{R}^{l' \times m}$*,* $\mathbf{Q} \in \mathbb{R}^{l' \times n}$*, and an invertible matrix* $\mathbf{P} \in \mathbb{R}^{l' \times l'}$*, such that*

$$\mathbf{M}_i^\top \mathbf{N}_j = (\mathbf{P}^{-i\top}\mathbf{R})^\top (\mathbf{P}^j \mathbf{Q}), \tag{8}$$

Proof is given in Appendix. Although this proposition does not impose any additional constraint on $\mathbf{M}_i$ and $\mathbf{N}_j$, it suggests that effectively we only need to consider the case that

$$\mathbf{M}_i = \mathbf{P}^{-i\top}\mathbf{R}, \mathbf{N}_j = \mathbf{P}^j \mathbf{Q} \tag{9}$$

3.3 PermuteFormer

Based on the analysis of the previous section, we introduce PermuteFormer by selecting specific $\mathbf{P}, \mathbf{Q}, \mathbf{R}$ in Equation 9.

To meet constraints imposed by Property 2 and Property 3, we choose the following solution for PermuteFormer.

$$\mathbf{R} = \mathbf{Q} = \mathbf{I}, \mathbf{P} = r^{-1}\mathbf{P}_\pi, \tag{10}$$

where $r = 1$ for bidirectional models and $0 < r < 1$ for unidirectional models, $\pi : \{1, 2, \cdots, m\} \to \{1, 2, \cdots, m\}$ is a permutation and $\mathbf{P}_\pi$ is the corresponding permutation matrix. (A permutation matrix is a square binary matrix that has exactly one entry of 1 in each row and each column and 0s elsewhere. For permutation π the corresponding permutation matrix $\mathbf{P}_\pi$ is the matrix that $\mathbf{P}_{\pi,ij} = 1$ if $\pi(i) = j$; $\mathbf{P}_{\pi,ij} = 0$ otherwise.) Note that different attention heads may have different $\mathbf{P}_\pi$ and r, so that both long-term and short-term dependencies are captured.

Substitute Equation 9, 10 into Equation 7, we get the similarity function of PermuteFormer

$$\mathrm{sim}_{\mathrm{Perm}}(\mathbf{q}_i, \mathbf{k}_j) = \left(r^i \mathbf{P}_\pi^i \phi(\mathbf{q}_i)\right)^\top \left(r^{-j}\mathbf{P}_\pi^j \phi(\mathbf{k}_j)\right). \tag{11}$$

PermuteFormer can encode relative positions up to the order of the permutation π. Goh and Schmutz (1991) proves that the order of random permutation grows exponentially with the head size. For a model with the same size as BERT-base (Devlin et al., 2019), the dimension of queries / keys per attention head is 64, corresponding to an average order of over 3000. To further extend PermuteFormer's ability to encode long sequences, we choose different permutations for different attention heads, so that the longest distance PermuteFormer can encode is the least common multiple of all permutations' orders, which can be up to 1e27 for a model with head size of 64.

There are two additional parameters PermuteFormer introduces, π and r. As π is a discrete parameter that cannot be optimized by gradient-based methods, we treat it as a hyper-parameter of the model. We randomly sample π at initialization of the neural network and fix its value during the whole training process. Although the model may get a better performance on training π, we find that a random permutation is good enough for PermuteFormer to work, so we do not tune π to save energy. Parameter r, on the other hand, can be optimized by gradient-based methods, but we also treat it as a hyper-parameter.

3.4 Computational Cost

We analyze computational cost of PermuteFormer in this section. PermuteFormer is as fast as Performer, which is the most efficient Transformer (Tay et al., 2020b) to our knowledge.

Let L denote the length of the sequence, H denote the number of heads in the model, and m denote the per-head hidden dimension of query features and key features.

The computational overhead introduced by PermuteFormer includes the computation of $\mathbf{P}_\pi^i$, the application of linear transformation $\mathbf{P}_\pi^i$ on query features and key features, as well as calculation of powers of r.

Multiplication of permutation matrices is equivalent to multiplication of corresponding permutations. In our case, it reads that

$$\mathbf{P}_\pi^i = \mathbf{P}_{\pi^i}, \tag{12}$$

where π^i is the i-th power of permutation π that

$$\pi^i(x) = \pi(\pi^{i-1}(x)) \text{ and } \pi^0(x) = x. \tag{13}$$

We can compute these π^i and cache them before training and inference. This takes $O(LHm)$ time and $O(LHm)$ memory.

As $\mathbf{P}_\pi^i$ is a permutation matrix, there is no need to do cumbersome matrix-vector multiplication. Instead, a `gather` operation on query features and key features is enough. The memory and time complexity of this `gather` operation is equal to the size of query features and key features, i.e., $O(LHm)$.

Powers of scalar r can be calculated easily.

Thus, the total overhead introduced by PermuteFormer is $O(LHm)$. Since the complexity of attention in Performer is $O(LHm^2)$, this overhead is negligible.

3.5 Trick for Two-Dimensional Case

As Transformer-based models are getting popular in fields other than natural language processing these days, it is worth noting that PermuteFormer is also applicable to 2D inputs like images and multi-modal documents (Xu et al., 2020).

One naive way to deal with two-dimension inputs is to follow the convention in benchmark Tay et al. (2020b). Pixels in the 2D space are first flattened to an 1D sequence before fed into the model. However, this causes problems for relative position encoding. It makes the rightmost pixel in the first row adjacent to the leftmost pixel in the second row, so the relative position of these two distant pixels is extremely close in the 1D sequence, which is incorrect. It is almost impossible for the model to learn something meaningful out of the wrong relative position.

To remedy this, we adapt PermuteFormer's attention for 2D inputs. We permute some elements of the query / key feature according to a pixel's horizontal position, while others according to its vertical position. More precisely, we modified equation 11 as follows

$$\begin{aligned}\mathrm{sim}_{\mathrm{Perm}}(\mathbf{q}_i, \mathbf{k}_j) = \big(\mathbf{P}_{\pi_x}^{x_i}\mathbf{P}_{\pi_y}^{y_i}\boldsymbol{\phi}(\mathbf{q}_i)\big)^\top \\ \big(\mathbf{P}_{\pi_x}^{x_j}\mathbf{P}_{\pi_y}^{y_j}\boldsymbol{\phi}(\mathbf{k}_j)\big),\end{aligned} \tag{14}$$

where (x_i, y_i), (x_j, y_j) are coordinates of the i-th and j-th pixel, respectively. π_x and π_y are two permutations commutative with each other.

4 Experiments

We evaluate bidirectional PermuteFormer on Long-Range Arena, which consists of many long-sequence tasks. Unidirectional PermuteFormer is evaluated on WikiText-103, a language modeling task.[3]

4.1 Long-Range Arena

Long-Range Arena (Tay et al., 2020b) is a benchmark for efficient Transformers. It concentrates on efficient Transformers' performance on long sequences. The benchmark consists of five subtasks from various domains: byte-level text classification, byte-level document retrieval, image classification on sequence of pixels, Pathfinder, and long ListOps. We follow the evaluation protocol of Tay et al. (2020b), except that we exclude the long ListOps task from the benchmark, because a simple classifier on the first token[4] performs on par with the best model reported in Tay et al. (2020b). In the four selected tasks, image classification has 10 labels, while the others are binary classification tasks.

4.1.1 Setup and Implementations

We compare our PermuteFormer with the vanilla Transformer and Performer. A version of Su et al. (2021) is also implemented on Performer for comparison. In addition, we also list performances of other efficient Transformers from Xiong et al. (2021), including Reformer (Kitaev et al., 2020), Linformer (Wang et al., 2020) and Nyströmformer (Xiong et al., 2021). Conventional relative position encoding, such as Shaw et al. (2018); Dai et al. (2019); Raffel et al. (2020), is not included, as it is

[3] Long-Range Arena can be fetched from https://github.com/google-research/long-range-arena. WikiText-103 can be fetched from https://s3.amazonaws.com/research.metamind.io/wikitext/wikitext-103-v1.zip.

[4] This classifier outputs `0` if the first token is `[MIN`, outputs `9` if the first token of the sequence is `[MAX`, and outputs `4` otherwise. It achieves an accuracy of 37.25 on the test set.

Model	Text	Retrieval	Image	Pathfinder	Average
Transformer	63.99_{23}	80.02_{26}	42.83_{142}	72.40_{165}	64.81_{55}
w/ sinusoidal pos. emb.	64.06_{17}	79.81_{33}	**43.30**$_{152}$	**73.00**$_{183}$	65.04_{60}
Reformer	64.88	78.64	43.29	69.36	64.04
Linformer	55.91	79.37	37.84	67.60	60.18
Nyströmformer	65.52	79.56	41.58	70.94	64.40
Performer	63.95_{42}	79.82_{30}	43.08_{174}	72.63_{108}	64.87_{53}
RoFormer	**66.00**$_{17}$	75.27_{55}	26.16_{109}	58.87_{26}	56.57_{31}
PermuteFormer	65.95_{26}	**80.66**$_{26}$	43.02_{52}	72.91_{100}	**65.64**$_{30}$
w/o 2D rel. pos.	65.95_{26}	**80.66**$_{26}$	36.10_{95}	65.72_{59}	62.10_{29}
w/o Property 3	50.27	70.62	10.00	50.05	45.24

Table 1: Performance on Long-Range Arena in accuracy. Results of Transformer, Performer, RoFormer and all variants of PermuteFormer are evaluated by us. Results for Reformer, Linformer and Nyströmformer are taken from Xiong et al. (2021). Numbers reported by us are average accuracies of five runs. Standard deviations are shown as subscripts, in units of 0.01.

Model	Text (4K)	Retrieval (4K)	Image (1K)	Pathfinder (1K)
Transformer	622 (1.00×)	2404 (1.00×)	26 (1.00×)	180 (1.00×)
Reformer	437 (0.70×)	1086 (0.45×)	30 (1.15×)	153 (0.85×)
Linformer	323 (0.52×)	483 (0.20×)	14 (0.54×)	68 (0.38×)
Nyströmformer	332 (0.53×)	566 (0.24×)	15 (0.58×)	65 (0.36×)
Performer	354 (0.57×)	553 (0.23×)	13 (0.50×)	61 (0.34×)
PermuteFormer	361 (0.58×)	550 (0.23×)	13 (0.50×)	62 (0.34×)
Performer + T5-style pos. emb. (estimated)	28585 (46.0×)	81070 (33.7×)	3697 (142×)	24601 (137×)

Table 2: Training time for one epoch in seconds. Ratio to Transformer is included in parentheses. Lower is better. The sequence length of the first two tasks is 4000, while that of the last two is 1024.

Model	Text (4K)	Retrieval (4K)	Image (1K)	Pathfinder (1K)
Transformer	72.03 (1.00×)	20.23 (1.00×)	3.28 (1.00×)	3.89 (1.00×)
Reformer	30.60 (0.42×)	13.57 (0.67×)	13.44 (4.10×)	13.49 (3.47×)
Linformer	19.82 (0.28×)	4.33 (0.21×)	3.31 (1.01×)	4.20 (1.08×)
Nyströmformer	23.55 (0.33×)	9.28 (0.46×)	7.01 (2.14×)	9.14 (2.35×)
Performer	30.20 (0.42×)	5.09 (0.25×)	3.01 (0.92×)	3.80 (0.98×)
PermuteFormer	31.18 (0.43×)	5.18 (0.26×)	2.99 (0.91×)	3.89 (1.00×)

Table 3: Inference latency for one sample in milliseconds. Ratio to Transformer is included in parentheses. Lower is better. The sequence length of the first two tasks is 4000, while that of the last two is 1024.

almost computational infeasible to apply them to such long sequences.

For efficiency, we choose a simple feature map

$$\phi(\mathbf{x}) = \max(\mathbf{x}, \mathbf{0}) + \epsilon, \quad (15)$$

for both Performer and PermuteFormer. ϵ is added to the features to ensure that the denominator in Equation 2 is not zero. We set $\epsilon = 0.001$.

In this paper, all neural networks are trained from scratch. Learning rates are manually tuned on Transformer to match the results reported by other papers. Then, these hyper-parameters are fixed on training of Performer and PermuteFormer. Model sizes are the same as those described in Tay et al. (2020b). The hidden dimension of query features and key features are four times of that of queries and keys. Absolute position embedding is disabled for PermuteFormer. Models are optimized with Adam (Kingma and Ba, 2015). More details of hyper-parameters can be found in the appendix. Each experiment is run five times and the average accuracy is reported. Experiments are done on machines with 8 V100 GPUs.

4.1.2 Results

Performance The results are summarized in Table 1. It shows that the relative position encoding in PermuteFormer significantly improves the performance of Performer in all the tasks, including both language tasks and vision tasks. It not only achieves better accuracy than existing efficient Transformers without relative position encoding, but also performs better than vanilla Transformer, as well as Performer with Su et al. (2021)'s relative position encoding.

Efficiency We record the training time of each model on all the tasks, as well as their latency on inference. The result is listed in Table 2 and Table 3. It shows that Performer runs around two to three times faster than Transformer. The second line and the third line of the table indicate that PermuteFormer's speed is almost the same as that of Performer. This aligns with our analysis in § 3.4 that the overhead of PermuteFormer is negligible compared to the computation cost of Performer itself.

We take T5 (Raffel et al., 2020) as an example to illustrate that existing relative position encoding is computationally infeasible for long sequences. We train Performer with T5 with a few iterations to estimate the running time for one epoch. The result is shown in the last line of Table 2. It indicates that T5 is significantly slower than Transformer, not to say Performer.

Model	PPL
Transformer(Vaswani et al., 2017)	30.18
Performer(Choromanski et al., 2020)	36.87
PermuteFormer	32.49
PermuteFormer w/o r	35.76
PermuteFormer w/o $\mathbf{P}_\pi$	33.08

Table 4: Perplexity (PPL) of models on test split of WikiText-103 language modeling dataset.

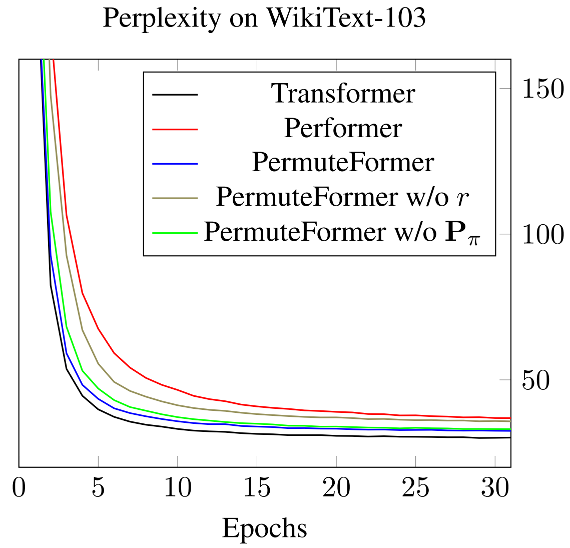

Figure 2: Trends of perplexity during training on test split of WikiText-103 language modeling dataset.

4.1.3 Ablation Study

We evaluate whether 2D relative position encoding is useful for PermuteFormer. We train PermuteFormer with 1D relative position encoding, and the result is shown in the second last line of Table 1. As expected, its performance drops significantly for tasks with 2D inputs. Thus, 1D relative position encoding is harmful to vision tasks as discussed in § 3.5.

We also justify that Property 3 is necessary for PermuteFormer, i.e., the transformation should preserve positiveness of query features and key features. We train a PermuteFormer with the permutation matrix $\mathbf{P}_\pi$ replaced by a random orthogonal matrix. The result is listed in the last line of Table 1, that PermuteFormer without Property 3 does not converge on most of the tasks.

4.2 WikiText-103

We evaluate unidirectional PermuteFormer on WikiText-103 (Merity et al., 2017). It is a language modeling dataset with about 103 million tokens extracted from verified articles on Wikipedia.

4.2.1 Setup and Implementations

We compare PermuteFormer with the vanilla Transformer and Performer. Models are implemented with `fairseq` (Ott et al., 2019). We adopt hyperparameters suggested by `fairseq`[5]: 6 layers, hidden dimension of 512, feed forward dimension of 1024, 8 attention heads. Feature map is the same as Equation 5. r takes its value in $[0.88, 0.99]$. For comparison with absolute position encoding, we set the sequence length to 512. Perplexity is measured on the test set. To avoid predicting tokens with little context at the beginning of a sequence, only the last 256 tokens are counted in the results. Effects of r and $\mathbf{P}_\pi$ are measured separately through ablation studies, i.e., removing r or $\mathbf{P}_\pi$ in Equation 11.

4.2.2 Results

The results for WikiText-103 are listed in Table 4. We also plot trending of perplexity during training in Figure 2. It shows that PermuteFormer lowers the performance gap between Transformer and Performer. It also speeds up convergence of models.

The last two lines of Table 4 indicate that performance of PermuteFormer drops without r or $\mathbf{P}_\pi$. Thus, both r and $\mathbf{P}_\pi$ are crucial for PermuteFormer. r may be helpful for PermuteFormer to focus on local context, while $\mathbf{P}_\pi$ is responsible for encoding relative positional information.

5 Conclusions

We discuss possible ways to add relative position encoding to Performer, a family of efficient Transformers scales linearly. Based on the analysis, we propose PermuteFormer, a variant of Performer with position-aware permutation to encode relative positional information. While improving the performance, this novel relative position encoding introduces negligible overhead compared to the overall computational cost of Performer. Experiments show that it runs as fast as Performer.

Extensive experiments are conducted on PermuteFormer, including byte-level text tasks and pixel-level image classification of Long-Range Arena, as well as language modeling on WikiText-103. Bidirectional PermuteFormer is used for the former tasks, while unidirectional PermuteFormer is adopted for the latter one. Results show that PermuteFormer uniformly improves the performance of Performer, accelerates convergence, and achieves state-of-the-art on some tasks.

Ethical Considerations

This paper does not introduce new datasets. All the experiments and discussions are based on public datasets, which have been widely used for years. This paper focuses on speeding up NLP models generally. It is not directly connected to specific real-world applications.

The purpose of this paper is to reduce the computational cost of Transformer without performance drop. We hope our work will reduce energy consumption for future work of NLP. We also try our best to reduce carbon cost in experiments, such as minimizing hyper-parameter tuning. It takes about 10 days on 8 V100 GPUs to get all the figures in this paper.

[5] We use the same command-line options as described in `https://github.com/pytorch/fairseq/tree/master/examples/language_model`.

References

Alexei Baevski, Yuhao Zhou, Abdelrahman Mohamed, and Michael Auli. 2020. wav2vec 2.0: A framework for self-supervised learning of speech representations. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *CoRR*, abs/2004.05150.

Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamás Sarlós, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, David Belanger, Lucy Colwell, and Adrian Weller. 2020. Rethinking attention with performers. *CoRR*, abs/2009.14794.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc Le, and Ruslan Salakhutdinov. 2019. 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, Florence, Italy. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. 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, Minneapolis, Minnesota. Association for Computational Linguistics.

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. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *CoRR*, abs/2010.11929.

William M. Y. Goh and Eric Schmutz. 1991. The expected order of a random permutation. *Bulletin of the London Mathematical Society*, 23(1):34–42.

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2020. Deberta: Decoding-enhanced BERT with disentangled attention. *CoRR*, abs/2006.03654.

Jungo Kasai, Hao Peng, Yizhe Zhang, Dani Yogatama, Gabriel Ilharco, Nikolaos Pappas, Yi Mao, Weizhu Chen, and Noah A. Smith. 2021. Fine-tuning pretrained transformers into rnns. *CoRR*, abs/2103.13076.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. 2020. Transformers are rnns: Fast autoregressive transformers with linear attention. *CoRR*, abs/2006.16236.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

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.

Valerii Likhosherstov, Krzysztof Choromanski, Jared Davis, Xingyou Song, and Adrian Weller. 2020. Sub-linear memory: How to make performers slim. *CoRR*, abs/2012.11346.

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.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. 2019. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, pages 48–53, Minneapolis, Minnesota. Association for Computational Linguistics.

Hao Peng, Nikolaos Pappas, Dani Yogatama, Roy Schwartz, Noah A. Smith, and Lingpeng Kong. 2021. Random feature attention. *CoRR*, abs/2103.02143.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21:140:1–140:67.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. 2018. Self-attention with relative position representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 464–468, New Orleans, Louisiana. Association for Computational Linguistics.

Jianlin Su, Yu Lu, Shengfeng Pan, Bo Wen, and Yunfeng Liu. 2021. Roformer: Enhanced transformer with rotary position embedding. *CoRR*, abs/2104.09864.

Hao Tan and Mohit Bansal. 2019. LXMERT: Learning cross-modality encoder representations from transformers. 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 5100–5111, Hong Kong, China. Association for Computational Linguistics.

Yi Tay, Dara Bahri, Donald Metzler, Da-Cheng Juan, Zhe Zhao, and Che Zheng. 2020a. Synthesizer: Rethinking self-attention in transformer models. *CoRR*, abs/2005.00743.

Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. 2020b. Long range arena: A benchmark for efficient transformers. *CoRR*, abs/2011.04006.

Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. 2020c. Efficient transformers: A survey. *CoRR*, abs/2009.06732.

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 *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008.

Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. 2020. Linformer: Self-attention with linear complexity. *CoRR*, abs/2006.04768.

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. *CoRR*, abs/2102.03902.

Yiheng Xu, Minghao Li, Lei Cui, Shaohan Huang, Furu Wei, and Ming Zhou. 2020. Layoutlm: Pre-training of text and layout for document image understanding. In *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*, pages 1192–1200. ACM.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 5754–5764.

Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontañón, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. 2020. Big bird: Transformers for longer sequences. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

A Proof of Proposition 1

Lemma A.1. *Let*

$$\{\mathbf{M}_i\}_{i=-\infty}^{\infty} \in \mathbb{R}^{l\times m}, \tag{16}$$

$$\{\mathbf{N}_i\}_{i=-\infty}^{\infty} \in \mathbb{R}^{l\times n}. \tag{17}$$

Assume

$$\forall i,j,k \in \mathbb{Z}, \mathbf{M}_i^\top \mathbf{N}_j = \mathbf{M}_{i+k}^\top \mathbf{N}_{j+k}. \tag{18}$$

Then, there exists

$$\{\mathbf{M}'_i\}_{i=-\infty}^{\infty} \in \mathbb{R}^{l'\times m'}, \tag{19}$$

$$\{\mathbf{N}'_i\}_{i=-\infty}^{\infty} \in \mathbb{R}^{l'\times n'}, \tag{20}$$

$$\mathbf{P} \in \mathbb{R}^{m'\times m}, \mathbf{Q} \in \mathbb{R}^{n'\times n}, \tag{21}$$

such that

$$\forall i,j,k \in \mathbb{Z}, \mathbf{M'}_i^\top \mathbf{N}'_j = \mathbf{M'}_{i+k}^\top \mathbf{N}'_{j+k}, \tag{22}$$

$$\forall i,j \in \mathbb{Z}, \mathbf{M}_i^\top \mathbf{N}_j = (\mathbf{M}'_i\mathbf{P})^\top(\mathbf{N}'_j\mathbf{Q}), \tag{23}$$

$$\sum_{i=-\infty}^{\infty} \operatorname{im}(\mathbf{M}'_i) = \mathbb{R}^{l'}, \sum_{i=-\infty}^{\infty} \operatorname{im}(\mathbf{N}'_i) = \mathbb{R}^{l'}, \tag{24}$$

$$\forall i \in \mathbb{Z}, \ker(\mathbf{M}'_i) = \{\mathbf{0}\}, \ker(\mathbf{N}'_i) = \{\mathbf{0}\}. \tag{25}$$

Proof. Induction on l, m, n.

If $l = m = n = 0$, then $\mathbf{M}'_i = \mathbf{M}_i$, $\mathbf{N}'_i = \mathbf{N}_i$, $\mathbf{P} = \mathbf{I}_m$, $\mathbf{Q} = \mathbf{I}_n$ satisfies Equation 22-25.

Obviously, $\mathbf{M}'_i = \mathbf{M}_i$, $\mathbf{N}'_i = \mathbf{N}_i$, $\mathbf{P} = \mathbf{I}_m$, $\mathbf{Q} = \mathbf{I}_n$ satisfies Equation 22-23.

Case 1) It does not satisfy Equation 24. Without loss of generality, assume $\sum_{i=-\infty}^{\infty} \operatorname{im}(\mathbf{M}_i) \neq \mathbb{R}^l$. Then $(\sum_{i=-\infty}^{\infty} \operatorname{im}(\mathbf{M}_i))^\perp \neq \{\mathbf{0}\}$. Let unit vector $\mathbf{x} \in (\sum_{i=-\infty}^{\infty} \operatorname{im}(\mathbf{M}_i))^\perp$. For any i, $\mathbf{x} \in \operatorname{im}(\mathbf{M}_i)^\perp$, so $\mathbf{M}_i = (\mathbf{I}_l - \mathbf{x}\mathbf{x}^\top)\mathbf{M}_i$. Since $\operatorname{rank}(\mathbf{I}_l - \mathbf{x}\mathbf{x}^\top) = l - 1$, there is $\mathbf{A} \in \mathbb{R}^{(l-1)\times l}$ such that $\mathbf{I}_l - \mathbf{x}\mathbf{x}^\top = \mathbf{A}^\top\mathbf{A}$.

Let $\tilde{\mathbf{M}}_i = \mathbf{A}\mathbf{M}_i$, $\tilde{\mathbf{N}}_i = \mathbf{A}\mathbf{N}_i$. Then, $\mathbf{M}_i^\top\mathbf{N}_j = \tilde{\mathbf{M}}_i^\top\tilde{\mathbf{N}}_j$. By induction, there is $\mathbf{M}'_i$, $\mathbf{N}'_i$, $\mathbf{P}$, $\mathbf{Q}$ that satisfies Equation 22-25.

Case 2) It satisfies Equation 24, but not Equation 25. Without loss of generality, assume unit vector $\mathbf{x} \in \ker(\mathbf{N}_i)$ for some i. Then, for any j, k, $\mathbf{M}_k^\top\mathbf{N}_j\mathbf{x} = \mathbf{M}_{k+i-j}^\top\mathbf{N}_i\mathbf{x} = \mathbf{0}$. Thus, $\mathbf{N}_j\mathbf{x} \in \operatorname{im}(\mathbf{M}_k)^\perp$ for any k. Equivalently, $\mathbf{N}_j\mathbf{x} \in (\sum_{k=-\infty}^{\infty} \operatorname{im}(\mathbf{M}_k))^\perp$. By Equation 24, $\mathbf{N}_j\mathbf{x} = \mathbf{0}$. So $\mathbf{x} \in \ker(\mathbf{N}_j)$ for any j.

Therefore, for any $j \in \mathbb{Z}$, $\mathbf{N}_j = \mathbf{N}_j(\mathbf{I}_n - \mathbf{x}\mathbf{x}^\top)$. Since $\operatorname{rank}(\mathbf{I}_n - \mathbf{x}\mathbf{x}^\top) = n - 1$, there is $\mathbf{A} \in \mathbb{R}^{(n-1)\times n}$ such that $\mathbf{I}_n - \mathbf{x}\mathbf{x}^\top = \mathbf{A}^\top\mathbf{A}$.

Let $\tilde{\mathbf{M}}_i = \mathbf{M}_i$, $\tilde{\mathbf{N}}_i = \mathbf{N}_i\mathbf{A}^\top$. Then $\tilde{\mathbf{M}}_i^\top\tilde{\mathbf{N}}_j = \mathbf{M}_i^\top\mathbf{N}_j\mathbf{A}^\top = \mathbf{M}_{i+k}^\top\mathbf{N}_{j+k}\mathbf{A}^\top = \tilde{\mathbf{M}}_{i+k}^\top\tilde{\mathbf{N}}_{j+k}$. By induction we have $\tilde{\mathbf{M}}'_i$, $\tilde{\mathbf{N}}'_i$, $\tilde{\mathbf{P}}$, $\tilde{\mathbf{Q}}$ that

$$\forall i,j,k \in \mathbb{Z}, \tilde{\mathbf{M}}'^\top_i\tilde{\mathbf{N}}'_j = \tilde{\mathbf{M}}'_{i+k}{}^\top\tilde{\mathbf{N}}'_{j+k},$$

$$\forall i,j \in \mathbb{Z}, \tilde{\mathbf{M}}_i^\top\tilde{\mathbf{N}}_j = (\tilde{\mathbf{M}}'_i\tilde{\mathbf{P}})^\top(\tilde{\mathbf{N}}'_j\tilde{\mathbf{Q}}),$$

$$\sum_{i=-\infty}^{\infty} \operatorname{im}(\tilde{\mathbf{M}}'_i) = \mathbb{R}^{\tilde{l}'}, \sum_{i=-\infty}^{\infty} \operatorname{im}(\tilde{\mathbf{N}}'_i) = \mathbb{R}^{\tilde{l}'},$$

$$\forall i \in \mathbb{Z}, \ker(\tilde{\mathbf{M}}'_i) = \{\mathbf{0}\}, \ker(\tilde{\mathbf{N}}'_i) = \{\mathbf{0}\}.$$

So $\mathbf{M}'_i = \tilde{\mathbf{M}}'_i$, $\mathbf{N}'_i = \tilde{\mathbf{N}}'_i$, $\mathbf{P} = \tilde{\mathbf{P}}$, $\mathbf{Q} = \tilde{\mathbf{Q}}\mathbf{A}$ satisfies Equation 22-25.

Case 3) It satisfies both Equation 24 and Equation 25. Nothing to prove. □

Lemma A.2. *Let*

$$\{\mathbf{M}_i\}_{i=-\infty}^{\infty} \in \mathbb{R}^{l\times m}, \tag{26}$$

$$\{\mathbf{N}_i\}_{i=-\infty}^{\infty} \in \mathbb{R}^{l\times n}. \tag{27}$$

Assume

$$\forall i,j,k \in \mathbb{Z}, \mathbf{M}_i^\top\mathbf{N}_j = \mathbf{M}_{i+k}^\top\mathbf{N}_{j+k}, \tag{28}$$

$$\sum_{i=-\infty}^{\infty} \operatorname{im}(\mathbf{M}_i) = \mathbb{R}^l, \sum_{i=-\infty}^{\infty} \operatorname{im}(\mathbf{N}_i) = \mathbb{R}^l, \tag{29}$$

$$\forall i \in \mathbb{Z}, \ker(\mathbf{M}_i) = \{\mathbf{0}\}, \ker(\mathbf{N}_i) = \{\mathbf{0}\}. \tag{30}$$

Then, there exists

$$\{\mathbf{M}'_i\}_{i=-\infty}^{\infty} \in \mathbb{R}^{l\times l}, \tag{31}$$

$$\{\mathbf{N}'_i\}_{i=-\infty}^{\infty} \in \mathbb{R}^{l\times l}, \tag{32}$$

$$\mathbf{P} \in \mathbb{R}^{l\times m}, \mathbf{Q} \in \mathbb{R}^{l\times n}, \tag{33}$$

such that

$$\forall i,j,k \in \mathbb{Z}, \mathbf{M'}_i^{\top}\mathbf{N}'_j = \mathbf{M'}_{i+k}^{\top}\mathbf{N}'_{j+k}, \quad (34)$$

$$\forall i,j \in \mathbb{Z}, \mathbf{M}_i^{\top}\mathbf{N}_j = (\mathbf{M}'_i\mathbf{P})^{\top}(\mathbf{N}'_j\mathbf{Q}), \quad (35)$$

$$\sum_{i=-\infty}^{\infty} \mathrm{im}(\mathbf{M}'_i) = \mathbb{R}^l, \sum_{i=-\infty}^{\infty} \mathrm{im}(\mathbf{N}'_i) = \mathbb{R}^l, \quad (36)$$

$$\forall i \in \mathbb{Z}, \ker(\mathbf{M}'_i) = \{\mathbf{0}\}, \ker(\mathbf{N}'_i) = \{\mathbf{0}\}. \quad (37)$$

Proof. Induction on $l-m$, $l-n$.

If $l-m = l-n = 0$, then $\mathbf{M}'_i = \mathbf{M}_i$, $\mathbf{N}'_i = \mathbf{N}_i$, $\mathbf{P} = \mathbf{I}_l$, $\mathbf{Q} = \mathbf{I}_l$ satisfies Equation 34-37.

Without loss of generality, we only need to discuss the case that $n < l$.

If $n < l$, $\mathrm{im}(\mathbf{N}_0) \neq \mathbb{R}^l$. On the other hand, $\sum_{i=-\infty}^{\infty} \mathrm{im}(\mathbf{N}_i) = \mathbb{R}^l$. So there is a column of $\mathbf{N}_p$ for some $p \neq 0$ that in $\mathbb{R}^l \backslash \mathrm{im}(\mathbf{N}_0)$. More generally, there is a vector $\mathbf{e} \in \mathbb{R}^n$ and an integer p, that $\mathbf{N}_p\mathbf{e} \in \mathbb{R}^l \backslash \mathrm{im}(\mathbf{N}_0)$.

Let $\tilde{\mathbf{M}}_i = \mathbf{M}_i$, $\tilde{\mathbf{N}}_i = [\mathbf{N}_i, \mathbf{N}_{p+i}\mathbf{e}]$, $\mathbf{A} = [\mathbf{I}_n, \mathbf{0}_n]^{\top}$. Then,

$$\tilde{\mathbf{M}}_i^{\top}\tilde{\mathbf{N}}_j\mathbf{A} = \mathbf{M}_i^{\top}\mathbf{N}_j. \quad (38)$$

$$\begin{aligned} &\tilde{\mathbf{M}}_i^{\top}\tilde{\mathbf{N}}_j \\ =&[\mathbf{M}_i^{\top}\mathbf{N}_j, \mathbf{M}_i^{\top}\mathbf{N}_{p+j}\mathbf{e}] \\ =&[\mathbf{M}_{i+k}^{\top}\mathbf{N}_{j+k}, \mathbf{M}_{i+k}^{\top}\mathbf{N}_{p+j+k}\mathbf{e}] \\ =&\tilde{\mathbf{M}}_{i+k}^{\top}\tilde{\mathbf{N}}_{j+k}. \end{aligned} \quad (39)$$

$$\sum_{i=-\infty}^{\infty} \mathrm{im}(\tilde{\mathbf{M}}_i) = \sum_{i=-\infty}^{\infty} \mathrm{im}(\mathbf{M}_i) = \mathbb{R}^l. \quad (40)$$

$$\forall i \in \mathbb{Z}, \ker(\tilde{\mathbf{M}}_i) = \ker(\mathbf{M}_i) = \{\mathbf{0}\}. \quad (41)$$

$$\mathbb{R}^l \supset \sum_{i=-\infty}^{\infty} \mathrm{im}(\tilde{\mathbf{N}}_i) \supset \sum_{i=-\infty}^{\infty} \mathrm{im}(\mathbf{N}_i) = \mathbb{R}^l. \quad (42)$$

If for some i, $\ker(\tilde{\mathbf{N}}_i) \neq \{\mathbf{0}\}$, let $\mathbf{x}$ be a non-zero vector in $\ker(\tilde{\mathbf{N}}_i)$. Then, for any k, $\tilde{\mathbf{M}}_k^{\top}\tilde{\mathbf{N}}_0\mathbf{x} = \tilde{\mathbf{M}}_{k+i}^{\top}\tilde{\mathbf{N}}_i\mathbf{x} = \mathbf{0}$. Thus, $\tilde{\mathbf{N}}_0\mathbf{x} \in \mathrm{im}(\tilde{\mathbf{M}}_k)^{\perp}$ for any k. Equivalently, $\tilde{\mathbf{N}}_0\mathbf{x} \in (\sum_{k=-\infty}^{\infty} \mathrm{im}(\tilde{\mathbf{M}}_k))^{\perp} = \{\mathbf{0}\}$. So $\ker(\tilde{\mathbf{N}}_0) \neq \{\mathbf{0}\}$. However, by construction $\tilde{\mathbf{N}}_0 = [\mathbf{N}_0, \mathbf{N}_p\mathbf{e}]$, so $\ker(\tilde{\mathbf{N}}_0) = \{\mathbf{0}\}$. Thus,

$$\forall i \in \mathbb{Z}, \ker(\tilde{\mathbf{N}}_i) = \{\mathbf{0}\}. \quad (43)$$

By induction we have $\tilde{\mathbf{M}}'_i$, $\tilde{\mathbf{N}}'_i$, $\tilde{\mathbf{P}}$, $\tilde{\mathbf{Q}}$ that

$$\forall i,j,k \in \mathbb{Z}, \tilde{\mathbf{M}}_i'^{\top}\tilde{\mathbf{N}}'_j = \tilde{\mathbf{M}}_{i+k}'^{\top}\tilde{\mathbf{N}}'_{j+k},$$

$$\forall i,j \in \mathbb{Z}, \tilde{\mathbf{M}}_i^{\top}\tilde{\mathbf{N}}_j = (\tilde{\mathbf{M}}'_i\tilde{\mathbf{P}})^{\top}(\tilde{\mathbf{N}}'_j\tilde{\mathbf{Q}}),$$

$$\sum_{i=-\infty}^{\infty} \mathrm{im}(\tilde{\mathbf{M}}'_i) = \mathbb{R}^{\tilde{l}'}, \sum_{i=-\infty}^{\infty} \mathrm{im}(\tilde{\mathbf{N}}'_i) = \mathbb{R}^{\tilde{l}'},$$

$$\forall i \in \mathbb{Z}, \ker(\tilde{\mathbf{M}}'_i) = \{\mathbf{0}\}, \ker(\tilde{\mathbf{N}}'_i) = \{\mathbf{0}\}.$$

So $\mathbf{M}'_i = \tilde{\mathbf{M}}'_i$, $\mathbf{N}'_i = \tilde{\mathbf{N}}'_i$, $\mathbf{P} = \tilde{\mathbf{P}}$, $\mathbf{Q} = \tilde{\mathbf{Q}}\mathbf{A}$ satisfies Equation 34-37.

□

Proposition A.1. *Let $\{\mathbf{M}_i\}_{i=-\infty}^{\infty}$ be a series of $l \times m$ matrices, $\{\mathbf{N}_i\}_{i=-\infty}^{\infty}$ be a series of $l \times n$ matrices. Then, $\mathbf{M}_i^{\top}\mathbf{N}_j$ only depends on $i-j$, if and only if that, there is an integer l', matrices $\mathbf{P} \in \mathbb{R}^{l' \times m}$, $\mathbf{Q} \in \mathbb{R}^{l' \times n}$, and an invertible matrix $\mathbf{A} \in \mathbb{R}^{l' \times l'}$, such that*

$$\mathbf{M}_i^{\top}\mathbf{N}_j = (\mathbf{A}^{-i\top}\mathbf{P})^{\top}(\mathbf{A}^j\mathbf{Q}), \quad (44)$$

Proof. ($\Leftarrow$) *If* part.

$\mathbf{M}_i^{\top}\mathbf{N}_j = (\mathbf{A}^{-i\top}\mathbf{P})^{\top}(\mathbf{A}^j\mathbf{Q}) = \mathbf{P}^{\top}\mathbf{A}^{j-i}\mathbf{Q}$ depends on $i-j$ only.

($\Rightarrow$) *Only If* part.

By Lemma A.1 and Lemma A.2, there is

$$\{\mathbf{M}'_i\}_{i=-\infty}^{\infty} \in \mathbb{R}^{l' \times l'}, \quad (45)$$

$$\{\mathbf{N}'_i\}_{i=-\infty}^{\infty} \in \mathbb{R}^{l' \times l'}, \quad (46)$$

$$\mathbf{P} \in \mathbb{R}^{l' \times m}, \mathbf{Q} \in \mathbb{R}^{l' \times n}, \quad (47)$$

such that

$$\forall i,j,k \in \mathbb{Z}, \mathbf{M'}_i^{\top}\mathbf{N}'_j = \mathbf{M'}_{i+k}^{\top}\mathbf{N}'_{j+k}, \quad (48)$$

$$\forall i,j \in \mathbb{Z}, \mathbf{M}_i^{\top}\mathbf{N}_j = (\mathbf{M}'_i\mathbf{P})^{\top}(\mathbf{N}'_j\mathbf{Q}), \quad (49)$$

$$\forall i \in \mathbb{Z}, \mathrm{rank}(\mathbf{M}'_i) = l', \mathrm{rank}(\mathbf{N}'_i) = l'. \quad (50)$$

Since $\mathbf{M'}_0^{\top}\mathbf{N}'_{i-1} = \mathbf{M'}_1^{\top}\mathbf{N}'_i$,

$$\begin{aligned} \mathbf{N}'_i &= (\mathbf{M'}_1^{-\top}\mathbf{M'}_0^{\top})\mathbf{N}'_{i-1} & (51) \\ &= (\mathbf{M'}_1^{-\top}\mathbf{M'}_0^{\top})^i\mathbf{N}'_0 & (52) \\ &= \mathbf{A}^i\mathbf{N}'_0, & (53) \end{aligned}$$

where $\mathbf{A} \in \mathbb{R}^{l' \times l'}$ is an invertible matrix. Similarly, $\mathbf{M}'_i = \mathbf{B}^i\mathrm{M}'_0$. Substitute them into Equation 48, we have

$$\begin{aligned} &\forall i,j,k \in \mathbb{Z}, \\ &\mathbf{M'}_0^{\top}\mathbf{B}^{i\top}\mathbf{A}^j\mathbf{N}'_0 = \mathbf{M'}_0^{\top}\mathbf{B}^{(i+k)\top}\mathbf{A}^{j+k}\mathbf{N}'_0 \end{aligned} \quad (54)$$

Since A, B, $\mathbf{N}'_0$ and $\mathbf{M}'_0$ are invertible,

$$\forall ki \in \mathbb{Z}, \mathbf{B}^{k\top}\mathbf{A}^k = \mathbf{I}. \quad (55)$$

Thus, $\mathbf{B} = \mathbf{A}^{-\top}$.

Thus,

$$\begin{aligned} \mathbf{M}_i^{\top}\mathbf{N}_j &= (\mathbf{M}'_i\mathbf{P})^{\top}(\mathbf{N}'_j\mathbf{Q}) & (56) \\ &= (\mathbf{B}^i\mathbf{M}'_0\mathbf{P})^{\top}(\mathbf{A}^j\mathbf{N}'_0\mathbf{Q}) & (57) \\ &= (\mathbf{A}^{-i\top}\mathbf{P}')^{\top}(\mathbf{A}^j\mathbf{Q}'), & (58) \end{aligned}$$

where $\mathbf{P}' = \mathbf{M}'_0\mathbf{P}$ and $\mathbf{Q}' = \mathbf{N}'_0\mathbf{Q}$.

□

B Hyper-parameters for Long-Range Arena

Task	Text	Retrieval	Image	Pathfiner
Batch size	16	32	256	256
Epochs	10	10	50	80
LR	1e-5	2e-4	1e-2	5e-4
Warmup	4000	1000	200	4000

Table 5: Hyper-parameters for Long-Range Arena