Method for ViT Performance Optimization Based on Token Permutation and Spatial Reduction

Aleksei Kuchkov[1], Yaroslav Shubin[1], Alexey Kashevnik[2], Kseniia Ezhova[1], Larisa Gonchar[1], Ekaterina Brovkina[1], Svetlana Nerobova[1]
[1]ITMO University, St. Petersburg, Russia
[2]SPC RAS, St. Petersburg, Russia
kuchkovaleksei@gmail.com, irshubin@itmo.ru, alexey.kashevnik@iias.spb.su, ezhovakv@itmo.ru, lgonchar91@mail.ru, brovkina20146@gmail.com, glacons@rambler.ru

***Abstract*—Visual transformers (ViT) have several significant advantages over conventional architectures. Such advangtages include global context analysis and high scalability, which are particulary useful for classification, detection, and segmentation tasks. However, the core component of the transformer architecture is self-attention algorithm, which has large performance issues caused by its complexity $O(N^2)$ that requires performance optimization to use ViT for real world applications. In order to mitigate this issue we propose a method for self-attention approximation without quadratic cost, a parameter-free token permutation technique which can be used as an approximate replacement for a conventional self-attention. Our method has $O(N)$ computational cost. Moreover, we propose spatial reduction technique applied on the tokenization stage, which uses Fast Fourier transform to spatially compress token dimensions in half. Both techniques enable our method to run smoothly on embedded devices using either CPU or GPU accelerators in coomparison with baseline ViT and F-Net architectures. Implementation of our (SpecFormer) method was evaluated on CIFAR-100 dataset and demonstrated the significant parameter crop compared with alternative method with neglectable accuracy degradation.**

I. Introduction

Most of the modern SOTA computer vision models are build on top of the ViT architecture, which explores global relations between individual parts of an input data via attention mechanism. However, due to the high computational cost of the attention it becomes struggling to use transformers on embedded devices or expand into sparse modalities, such as point clouds. Moreover, base ViT architecture tends to overfit when training from scratch, which makes it complicated to use when dealing with small custom datasets. Thus, most of the current methods of feature extraction combine ViT in a hybrid architecture with CNN blocks, sparse convolution, and data serialization [1].

There are attempts to reduce complexity of attention and number of parameters in ViT architecture using spectral analysis. Methods, such as F-Net [2], replace attention layers with FFT transform, which mixes tokens with $O(nlogN)$ complexity instead of original $O(N^2)$. It was explored that various transformations, including Fourier Transform and Hadamard Transform can effectively mix tokens and serving as the substitute for Self-Attention.

Initially, spectral methods in deep neural networks were used for accelerating CNN inference and training [3], [4], [5], to improve transformation invariance [6], [7], and for audio recognition [8]. It was noticed that different spectral transformation methods enable faster convolution computation in spectral domain rather than in sparse domain. Moreover, depending on a particular transformation, it can represent global features at $Nlog(N)$ computational cost (FFT, Hadamard). The most novel approaches incorporate hierarchical patch-wise FFT to capture global-local relations in a transformer-like architecture [9].

In spite of the large progress in this domain, currently presented methods focus on large-transformer architectures, which are primarily running on GPU or TPU. Thus, there is a gap in low and mid-level model sizes executable on embedded devices using either CPU or NPU. Our motivation was to develop a method for creating models optimized for real-time inference on low-powered devices. Moreover, we aimed to reduce the complexetity of developed models in order to make them easely portable in various formats, such as ONNX [10], OpenVINO [11] and other formats.

Our contribution can be summarized as follows.

- We developed a substitution for linear layers, which can be integrated in various networks;
- We developed a light weight multi-head architecture, which performs token mixing without learnable parameters at $O(N)$ cost;
- We apply early-stage feature reduction based on FFT transformation during patch tokenization stage.

II. Related work

Since the original release of the transformer architecture, there was a significant effort in reducing the complexity of self-attention computation down to $Nlog(N)$ and lower.

One group of methods for faster self-attention computations include block-wise self-attention [12], multi-query attention [13], grouped-query attention (GQA) [14], sliding window attention (Longformer) [15], Flash Attention [16], and Paged Attention [17]. Block-wise attention reduces the size of the query matrix Q b grouping similar columns with locality-sensitive hashing (LSH). Multi-query attention use shared keys and values across different self-attention heads thereby reducing memory bandwidth requirements. Grouped-query

attention aims to handle issues caused by multi-query attention quality degradation by introducing intermediate key-value heads, which number is lower than number of self-attention heads. Sliding window attention, introduced in the Longformer architecture, utilizes fixed-size window with dilation to compute self-attention values without computing the whole N^2 matrix. Flash attention uses tiling to reduce the number of memory reads/writes between GPU high bandwidth memory (HBM) and GPU on-chip SRAM. Finally, paged attention builds a cache manager system to provide paging layout in analogous with virtual memory in operating systems. It enables Longformer to store continuous attention key and values in noncontinuous blocks in the memory. The provided methods, despite their effectiveness on GPU devices and with large-language models, are mostly specialized on exploiting limitations of CUDA memory layout, making them hardly transferable to other platforms, such as CPU, NPU, and TPU.

Another group of methods tries to reduce algorithmic complexity, targeting the initial problem. F-Net [2] was the first work where Fourier transform was used as a token mixing strategy instead of multi-head self-attention. Fast-FNet [18] went further by implementing the multistage Fourier transformation with real FFT, thereby reducing the encoder input parameters in half at each encoder layer. SpectFormer [19] continues the original F-Net idea while combining both conventional self-attention and spectral self-attention together and achieves SOTA results CIFAR-10, CIFAR-100, Oxford-IIIT-flower, and Standford Car datasets. SPECTRE [20] uses real FFT to obtain spectral features and gates them per frequency with a content-adaptive diagonal mask. Then, SPECTRE transforms spectral features back into the spatial domain with inverse real FFT. In [21] authors exploit orthogonality and energy preservation guaranteed by Parseval's theorem and apply both global FFT transform and windowed FFT.

In this work, we show that utilization the multi-head outputs of the image network improves knowledge distillation performance with initial problem. F-Net [9] was the first to notice token permutation advances when performing self-attention with the Fourier transform.

III. Background

The core component of the transformer architecture [22] is self-attention algorithm. Despite its advantages in token mixing, it consists of several $N \times N$ matrix multiplications. Self-attention calculates cosine similarity scores for each token in the sequence with other tokens. This process starts with calculating 3 matrices, Q - quires, K - keys, and V - values:

$$Q = XW_Q, K = XW_K, V = XW_V \tag{1}$$

where W_Q, W_K, and W_V – are weight matrices for Q, K, and V with the shape $d_{model} \times d_k$.

$$Y = Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2}$$

This process repeats for each individual head and follows by concatenation:

$$MH(Q, K, V) = Concat(head_1, ..., head_n)W_O \tag{3}$$

where W_O is the weight matrix for concatenated features.

The original self-attention implementation suffers from high computational cost due to many $O(N^2)$ operations. However, the core idea behind self-attention is token mixing, which can be performed in various ways with lower resource requirements. In F-Net [2] it was notices that Fast Fourier Transform can be used as a substitution for self-attention:

$$y = \mathcal{R}(\mathcal{F}_{seq}(\mathcal{F}_h(x))) \tag{4}$$

where $\mathcal{F}_{seq}$ and $\mathcal{F}_h$ are Fast Fourier transformations across token and embedded dimensions.

The actual performance of token mixing with FFT depends on the efficiency of FFT implementation. Usually, it performs fast on GPU and CPU as long as input dimensions are divisible by 2. However, there is an observed performance drop on Google TPU implementation [2].

Besides FFT, other transformations can be applied. Hadamard, for example, consist of $[-1, 1]$ coefficients and can be efficiently implemented with additions and subtractions:

$$H(u,v) = \frac{1}{N}\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} f(x,y)(-1)^{\sum_{i=0}^{n-1}[b_i(x)b_i(u)+b_i(y)b_i(v)]} \tag{5}$$

where $b_i(x)$ represents the i-th bit of x.

IV. Proposed Method

The core of our method is random permutation to token and embedding dimensions in the training stage. Once permutation is initialized, it becomes frozen and does not have trainable parameters. With the multi-head architecture, it enables us to apply different permutations per head, which evolves into distinct learned features. Due to its simplicity, permutation costs only $O(N)$. Another component of our method is the tokenization in the spectre domain. It allows us to reduce the spatial size of the image patches in half in the early execution stage with a single RFFT call, which adds a neglected computational cost. Fig. 1 represents the proposed method.

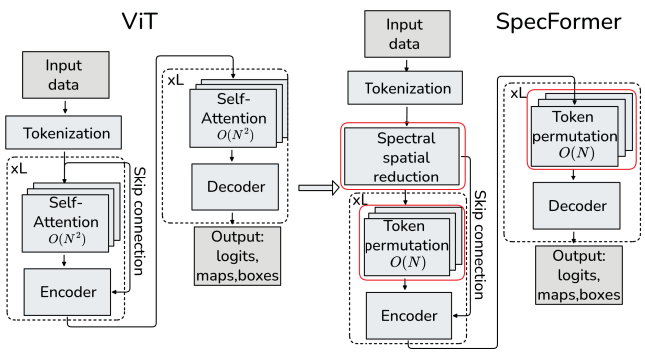


Fig. 1. The proposed method in comparison with the ViT (we highlighed red the proposed blocks for ViT enhancement)

The proposed permutation block can be used in both encoders and decoders, splitting a feature vector into H vectors, where H is the number of heads.

A. Spatial reduction in patch tokenization

There are several strategies to reduce spatial dimensions and number of parameters with spectre transformation. One of them is to apply real FFT transformation to inputs of each of the encoders as it was performed in Fast-FNet [18]. Another method can be implemented inside linear layers followed with RFFT call applied to the input and IRFFT to the output. We found out that the first approach can have an unpredictable performance issues caused by different FFT implementations in Deep Learning frameworks. For example, PyTorch **torch.fft.rfft** implementation can effectively pad an input shape to the next power of two if it is necessary. However, ONNX RFFT implementation suffers from performance drops when input is not perfectly aligned by power of 2. Moreover, FFT implementation shows worse performance in comparison with PyTorch module. Thus, we aimed to reduce number of FFT calls to minimum. The architecture and principal blocks for a classification task if shown on Fig. 2.

We applied 2D Real FFT (RFFT) transformation to the input tokens to shrink one of the spatial dimensions in half. It enables us to control spatial compression rate omitting high frequency bins from each individual patch. To improve generalizability of the model we use shared weights applied to patches right after Fourier Transform. The figure 3 represents the proposed spectre tokenization method. When applying spectre cutting at early stage we leave a burden of sustaining power of two feature invariant. Precisely, we apply 2D real FFT to each patch:

$$\bar{X}_{b,c}^{(n)} = \mathcal{F}_{rFFT}\left(X_{b,c}^{(n)}\right) \in \mathcal{R}^{P\times\left(\frac{P}{2}+1\right)} \tag{6}$$

where $X_{b,c}^{(n)}$ is a patch at n token, b batch and c channel. P is a patch size in pixels.

After applying $\mathcal{F}_{rFFT}$ transform to patches, we perform element-wise multiplication with shared weights $\mathrm{w}^h \in \mathcal{R}^P$ and $\mathrm{w}^w \in \mathcal{R}^{\frac{P}{2}+1}$ which introduce frequence importance scaling:

$$\bar{X}_{b,c,i,j}^{(n)} = \bar{X}_{b,c,i,j}^{(n)} \cdot W_{i,j} \tag{7}$$

where i and j are pixel coordinates in the patch n.

The resulted patches are then flattened into the shape:

$$s_b^{(n)} = vec\left(\bar{X}_{b,:,:,:}^{(n)}\right) \in \mathcal{R}^{N\times CP(\frac{P}{2}+1)} \tag{8}$$

where N is the number of tokens. Finally, we project each token into the embedding space:

$$z_b^{(n)} = W_e s_b^{(n)} + b_e \tag{9}$$

where $W_e \in \mathcal{R}^{D\times CP\left(\frac{P}{2}+1\right)}$ is the embedding transformation matrix and b_e is its bias.

As well as in the conventional ViT architecture, we add learned positional embeddings and cls token to the embeddings. The spectral patch embedding splitting is depicted on the Fig. 3.

B. Random token permutation

Our main goal was to approximate behavior of self-attention algorithm with lightweight architecture. We experiment with several approaches, including FFT transformation, Hadamard transformation, linear FFT approximation, and random permutations with sign flipping. In random permutation scenario for every head we permute tokens with randomly predefined indices and signs. For each head $h \in \{1, ..., H\}$ we define a permutation:

$$\pi_h : \{1, ..., D\} \Rightarrow \{1, ..., D\} \tag{10}$$

where D is embedding dimension multiplied by number of tokens. Also, we define a sign vector with the same size – D:

$$s_h \in \{-1, +1\}^D \tag{11}$$

With broadcast operations, we compute permuted features with shapes $R^{B\times H\times E\cdot N}$ where E – embedding dimension, N – number of tokens, and H is the number of heads. Then, we reshape permuted vector into the shape $Z \in R^{B\times N\times E\cdot H}$ and extract features with a single projection head:

$$Y = W \cdot Z + b \tag{12}$$

where $W \in R^{C_{out}\times\{C_{in}\cdot H\}}$ is the linear weight matrix for fused heads, and b – is the bias vector.

Fig. 4 represents the proposed token mixing strategy.

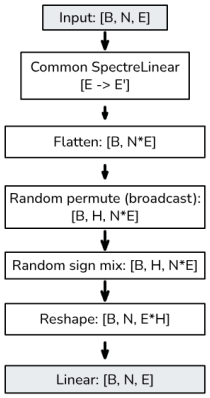


Fig. 4. Multi-head fused token mixing. B – batch size, N – number of tokens, E – embedding dimension, H – number of heads

Permutation indices and signs initialized at the training stage without further modifications. Our proposed multi-head permutation results in distinct features extracted by linear layer in the end of the permutation block. Although we do not perform global mixing as it was done in [2] and [18], we proved that simple sign mixing and feature permutation can be enough to imitate self-attention behavior without $O(N^2)$ complexity with acceptable accuracy drop.

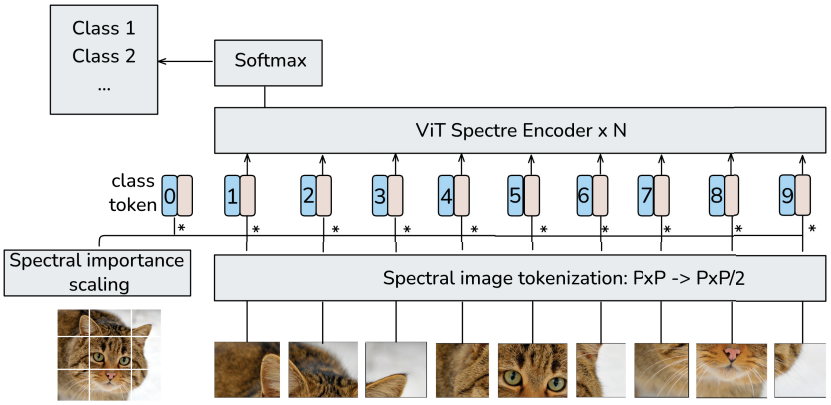


Fig. 2. Spectral image tokenization for the example of classification task

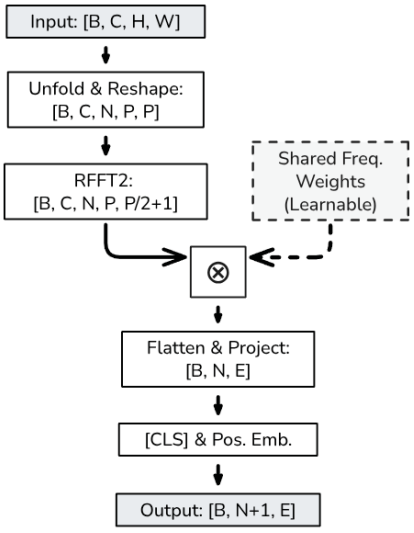


Fig. 3. Spatial reduction with RFFT transform

V. Experiments

We evaluated our method on MNIST [23] and CIFAR-100 datasets [24]. The choice of CIFAR-100 dataset was dictated by its simplicity as well as a variety of provided classes. We tested the following token mixing scenarios:

- F-Net architecture with FFT token mixing;
- Random permutation token mixing;
- Conventional Self-Attention.

We tested the performance of our method on both the CPU and GPU. Additionally, we evaluated the model inference speed on the Zynq UltraScale embedded platform with a 4xCortexA53 ARM CPU using the ONNX Runtime library.

Our experiments were conducted with the following conditions:

- Epochs – 500 (early stop, highest val score selection);
- Learning rate – 0.01, optimizer – AdamW, Batch – 8;
- Scheduler – Cosine;
- GPU: NVIDIA RTX 3080;
- CPU: AMD Ryzen 9 5900HX;
- Augmentations: Flip, Jitter, Gauss blur, Random Affine, Random affine, Random Mask.

A. Accuracy evaluation on MNIST and CIFAR-100

Table I represents results on MNIST dataset and Table II represents results on CIFAR-100 dataset. Tables provide a comparison of our method with conventional ViT and FNet, which uses FFT for self-attention. In addition, we provide an ablation analysis of the influence of patch size and head number on our network. The N/M notation represents N heads with M patch size (see SpecFormer N/M in the tables). FNet does not have heads and is denoted only by patch size M. Latency values are provided in milliseconds. In the MNIST dataset the accuracy is 95.1% vs 93.0% obtained on ViT (baseline) while reducing the inference latency by approximately 60% using only 105K parameters vs 202K (ViT). On the CIFAR-100 dataset, we achieved comparable validation performance with a 75% reduction in parameters (SpecFormer 8/4) compared to the baseline. Latency values are represented for batch size 8 on RTX 3080 (PyTorch inference). Tables III and IV demonstrate CortexA53 inference performance.

Table I. Experiment results on mnist dataset

Model	Hid	Enc	Weights	Lat.	Acc. (%)	
					Val	Train
ViT 8/4	256	4	202K	17.2	93.0	92.4
ViT 8/8	256	4	202K	17.2	91.3	92.3
FNet 4	256	4	151K	13.1	92.1	91.5
FNet 8	256	4	151K	13.1	91.0	91.4
SpecFormer 1/8 (Ours)	256	4	72K	8.17	88.2	87.6
SpecFormer 2/8 (Ours)	256	4	78K	8.13	88.9	88.2
SpecFormer 4/8 (Ours)	256	4	90K	8.18	90.8	90.1
SpecFormer 8/8 (Ours)	256	4	90K	8.09	91.8	91.2
SpecFormer 1/4 (Ours)	256	4	72K	8.58	88.5	87.9
SpecFormer 2/4 (Ours)	256	4	78K	8.59	91.3	90.6
SpecFormer 4/4 (Ours)	256	4	90K	8.34	90.5	89.8
SpecFormer 8/4 (Ours)	**256**	4	105K	6.57	**95.1**	**94.1**

TABLE II. EXPERIMENT RESULTS ON CIFAR-100 DATASET

Model	Hid	Enc	Weights	Lat.	Acc. (%)	
					Val	Train
ViT 8/4	256	4	4.85M	19.6	66.8	66.1
ViT 8/8	256	4	4.85M	17.9	67.2	66.4
FNet 4	256	4	3.62M	15.2	63.5	62.8
FNet 8	256	4	3.62M	13.8	64.1	63.3
SpecFormer 1/8 (Ours)	256	4	1.73M	8.95	58.4	57.6
SpecFormer 2/8 (Ours)	256	4	1.87M	9.02	60.2	59.4
SpecFormer 4/8 (Ours)	256	4	2.16M	9.11	63.7	62.9
SpecFormer 8/8 (Ours)	256	4	2.16M	9.26	65.5	64.6
SpecFormer 1/4 (Ours)	256	4	1.73M	10.8	59.3	58.5
SpecFormer 2/4 (Ours)	256	4	1.87M	11.0	64.2	63.4
SpecFormer 4/4 (Ours)	256	4	2.16M	11.3	66.1	65.3
SpecFormer 8/4 (Ours)	**256**	4	2.52M	11.6	**70.4**	**69.4**

TABLE III. MNIST EXPERIMENT RESULTS ON 4×CORTEX-A53 (ONNX RUNTIME CPU).

Model	Hid	Enc	Weights	Lat.	Acc. (%)	
					Val	Train
ViT 8/4	256	4	202K	242	93.0	92.4
ViT 8/8	256	4	202K	198	91.3	92.3
FNet 4	256	4	151K	186	92.1	91.5
FNet 8	256	4	151K	162	92.1	91.4
SpecFormer 1/8 (Ours)	256	4	72K	121	88.2	87.6
SpecFormer 2/8 (Ours)	256	4	78K	126	88.9	88.2
SpecFormer 4/8 (Ours)	256	4	90K	134	90.8	90.1
SpecFormer 8/8 (Ours)	256	4	90K	138	91.8	91.2
SpecFormer 1/4 (Ours)	256	4	72K	143	88.5	87.9
SpecFormer 2/4 (Ours)	256	4	78K	147	91.3	90.6
SpecFormer 4/4 (Ours)	256	4	90K	152	90.5	89.8
SpecFormer 8/4 (Ours)	**256**	4	105K	**150**	**95.1**	**94.1**

TABLE IV. CIFAR-100 EXPERIMENT RESULTS ON 4× CORTEX-A53 (ONNX RUNTIME CPU).

Model	Hid	Enc	Weights	Lat.	Acc. (%)	
					Val	Train
ViT 8/4	256	4	4.85M	268	66.8	66.1
ViT 8/8	256	4	4.85M	222	67.2	66.4
FNet 4	256	4	3.62M	208	63.5	62.8
FNet 8	256	4	3.62M	181	64.1	63.3
SpecFormer 1/8 (Ours)	256	4	1.73M	138	58.4	57.6
SpecFormer 2/8 (Ours)	256	4	1.87M	144	60.2	59.4
SpecFormer 4/8 (Ours)	256	4	2.16M	151	63.7	62.9
SpecFormer 8/8 (Ours)	256	4	2.16M	156	65.5	64.6
SpecFormer 1/4 (Ours)	256	4	1.73M	162	59.3	58.5
SpecFormer 2/4 (Ours)	256	4	1.87M	167	64.2	63.4
SpecFormer 4/4 (Ours)	256	4	2.16M	172	66.1	65.3
SpecFormer 8/4 (Ours)	**256**	4	**2.52M**	**158**	**70.4**	**69.4**

Experiments demonstrated that the proposed architecture maintains similar accuracy to baselines (ViT and F-Net) while significantly reducing model size and computational cost.

The deployment experiment evaluation on the ARM Cortex-A53 (ONNX Runtime CPU) embedded platform (Table III and Table IV) proves the practical viability of our method for edge computing applications. For MNIST dataset we compared with original baseline and got 150ms for the proposed SpecFormer 8/4 method vs 242 for ViT. For CIFAR-100 dataset we compared with original baseline and got that the proposed SpecFormer 8/4 method shows latency 158ms vs 268 for ViT. The ability to run efficiently on CPU without relying on specialized matrix multiplication accelerators makes this approach highly accessible for low-power devices.

B. Multi-head permutation performance evaluation

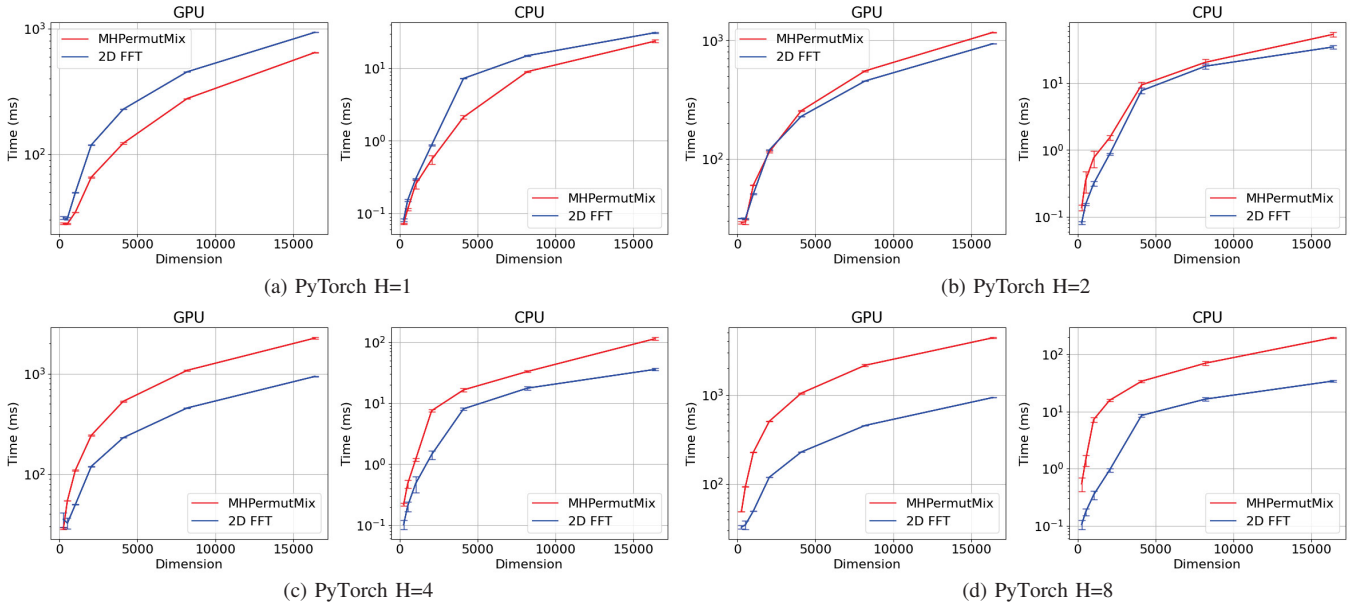
In this section, we investigated performance metrics for the core component of our method – multi-head random permutation with sign flipping (MHPermuteMix). We conducted an experiment with the following scenarious:

- Multi-head permutation performance on CPU and GPU – PyTorch runtime;
- Multi-head permutation performance on CPU – ONNX Runtime.

In order to provide comparison of our method with alternatives, we conducted tests with GPU and CPU for various embedding dimension sizes and number of heads. Additionally, we provided analysis of power of 2 input dimensions, and the dimensions, which do not satisfy this condition to investigate padding efficiency for FFT implementations in PyTorch (CPU and GPU) and ONNX CPU runtime.

We tested our multi-head permutation method in comparison with 2D FFT transform from F-Net [2]. We observed a difference in PyTorch and ONNX Runtime (CPU) implementations, which leads to ineffective inference performance of Fast Fourier Transform on embedded devices. While 2D FFT F-Net approach can perform faster on CPU when number of heads increases, it suffers from power of 2 requirements and ineffective implementations. Permutations, on the other hand, outperform 2D FFT asymptotically on both CPU and GPU. Moreover, the proposed MHPermuteMix imitates self-attention multi-head behavior with a low cost. Fig. 5 demonstrates the results obtained on PyTorch CPU and GPU in single-head and multi-head scenarios.

The performance of the observed method linearly depends on the number of heads on both the CPU and GPU. In ONNX CPU implementation the proposed permutation method outperforms 2D FFT in all provided scenarios. In PyTorch implementation, both methods have similar performance, the gap increasing as the number of heads grows. It was also noticed that unlike PyTorch, CPU Onnx runtime performs faster with DFT rather than FFT, which can significantly reduce performance if the size of the input features is not aligned to a power of 2. In our method, we apply FFT only once – at the image tokenization stage, which makes the choice between FFT and DFT insignificant. However, one may choose DFT because it can be run on almost any platform due to its implementation as the simple matrix multiplication. We investigated several runtimes in order to establish a limited set of operators, which we can use to achieve the highest transferability among different platforms.



(a) PyTorch H=1

(b) PyTorch H=2

(c) PyTorch H=4

(d) PyTorch H=8

Fig. 5. PyTorch multi-head permutation performance in comparison with 2D FFT (F-Net) token permutation

C. Multi-head weight distributions

We analysed weights of the first linear layers after our random permutation layer (the last linear layer in Fig. 4) for both MNIST and CIFAR100 datasets. Each column and layer represent a similarity score for a particular head with other heads, denoted as H_i where $i \in [1..8]$. Fig. 6 and 7 represent centered kernel alignment heatmaps for weights of the linear layers in each of the encoders. We split weights of the linear layers into vectors, where each one of them represents individual heads:

$$[C_{in}, C_{out} \times H] \Rightarrow [C_{in}, C_{out}]_i \mid i \in [1..H] \qquad (13)$$

where H – number of heads, C_{out} and C_{in} are output and input channels of the permutation block.

We observed that deeper layers (in the second and following encoders) tend to be more orthogonal to each other, which results in more distinctive feature maps obtained from different heads. Moreover, it seems to be that the higher entropy in the original image results in more orthogonal weights, which can be seen from comparision of MNIST and CIFAR100 heatmaps. The effect of the observed orthogonality and similarity variations should be further invistigated with larger datasets.

D. Support of spectral transforms in various backends

Of all the major libraries, Open Neural Network Exchange (ONNX) Runtime natively supports DFT and real-valued short-time FT (STFT) operators starting with the operator set (opset) 17 [25], [26]. Similarly, Intel OpenVINO IR also has explicit support listed exclusively for DFT and STFT operators, with the addition of inverse STFT (ISTFT) [27]. Google's TensorFlow Lite (TFLite) IR doesn't have any stable, documented builtin for spectral operations [28], although RFFT2D-style kernels exist in the codebase [29]. Nvidia's TensorRT would require using a custom plugin which leverages cuFFT calls to implement an FFT spectral layer, however, as with TFLite, no native support has ever been declared [30]. Naturally, runtimes such as vLLM or DeepSpeed-Inference, which focus specifically on attention-centric flows, have spectral support totally outside their scope.

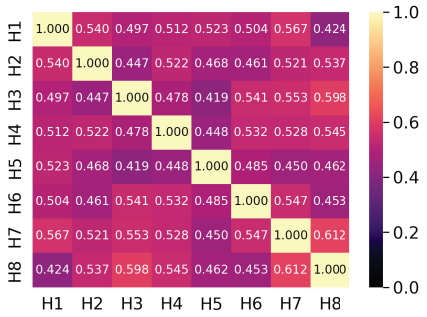
Considering the basically non-existent generalized spectral operator support at the current state of IR development, there exist only a couple of ways to integrate custom spectral layers:

- Leveraging PyTorch experimental *dynamo* export feature to integrate custom tensor-like discrete transforms with ONNX Runtime.
- Writing plugins/delegates for TensorRT/TFLite, respectively, manually adjusting hardware-specific limitations.
- Setting up a custom deployment stack using TorchServe, TorchScript and custom C++ plugins.

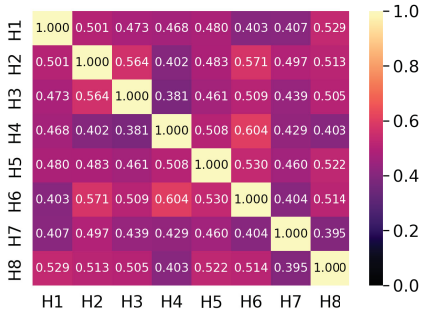
Among all options, only the custom TorchScript-based deployment flow would allow for maximum operator support, although it most definitely comes with additional challenges regarding hardware optimization. However, the proposed method can be easily integrated in various runtimes because it does not strictly relay on FFT, and it can be approximated with DFT without a significant performance drop.

VI. Discussion and Limitations

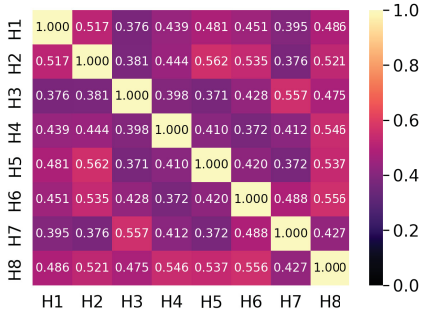
Despite the asymptotical effectiveness of the proposed method for the image classification task, it does not fully represent self-attention internal behavior. Instead of calculating cosine similarities between individual patches, it performs different shuffling orders for every head. In case of visual sequences, random permutation enables different weight distributions for every head. Thus, the performance of the proposed



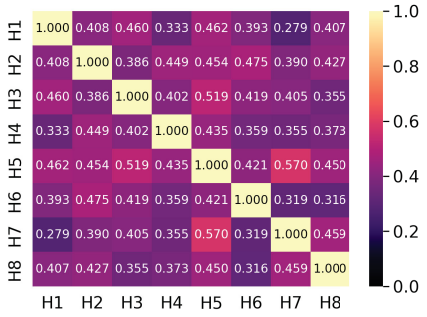
(a) Encoder 1



(b) Encoder 2

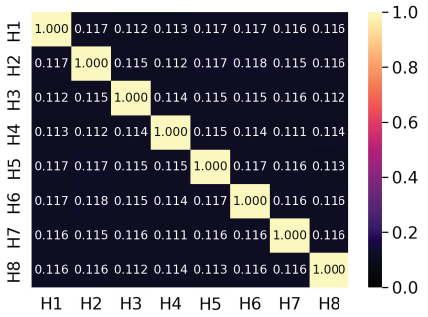


(c) Encoder 3

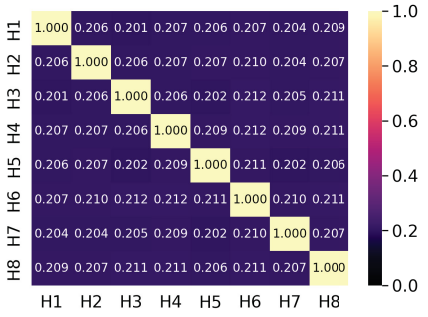


(d) Encoder 4

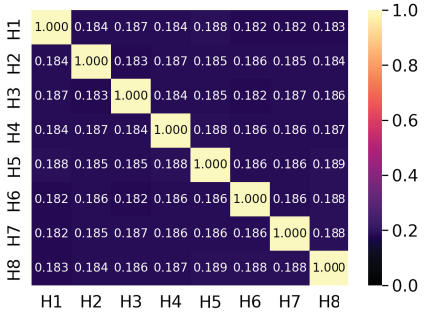
Fig. 6. Centered kernel alignment of the first linear layer after random permutation for MNIST dataset



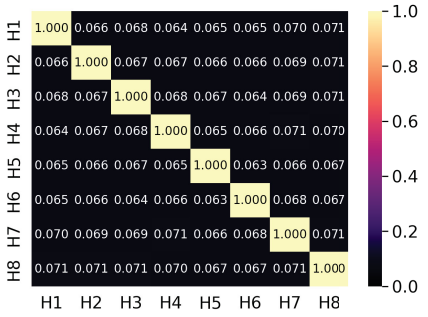
(a) Encoder 1



(b) Encoder 2



(c) Encoder 3

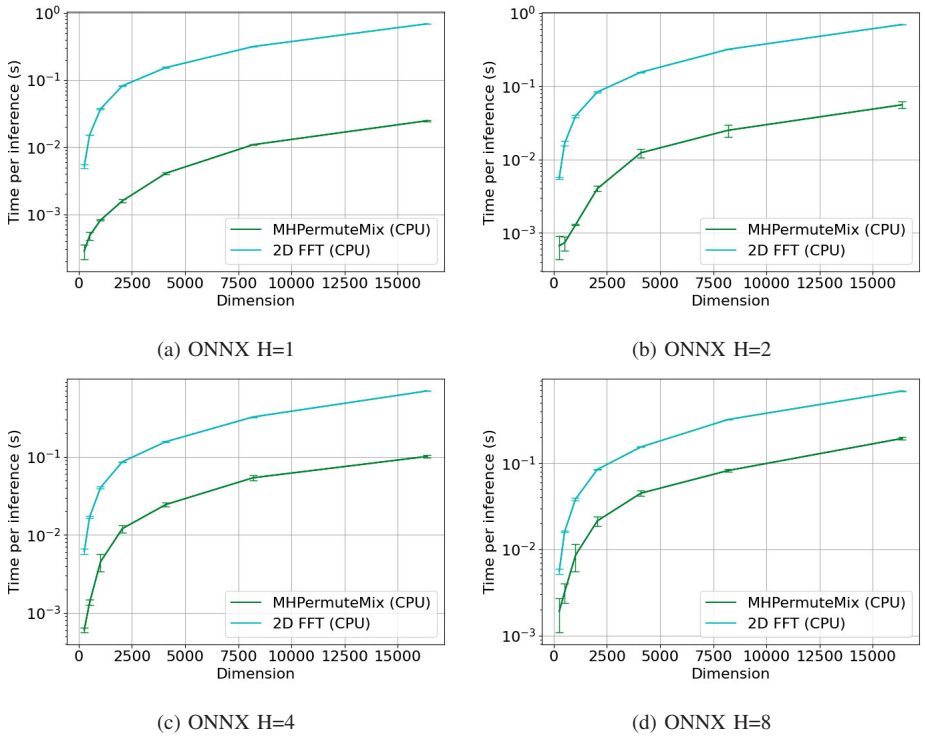


(d) Encoder 4

Fig. 7. Centered kernel alignment of the first linear layer after random permutation for CIFAR100 dataset

method should be further investigated in other deep learning tasks, such as NLP and audio processing. We assume that this method may not work well with long text sequences due to aggressive shuffling, which can lead to insufficient context understanding of natural language tokens. Moreover, a particular effect of heads orthogonality on accuracy metrics should be further investigated, as well as the similarity variations across different encoders. However, visual sequences gain an advantage due to spatial, time-localized structure of images.

Future work will focus on scaling this method to larger datasets, such as ImageNet [31], and exploring spectral transformation with 3D data, such as point clouds. That allows to compare the presented method with wider range of SOTA solutions (e.g., Swin Transformer, PVT). Also, we plan to



(a) ONNX H=1

(b) ONNX H=2

(c) ONNX H=4

(d) ONNX H=8

Fig. 8. ONNX multi-head permutation performance in comparison with 2D single-head FFT (F-Net) token permutation

investigate the relationship of accuracy and inference / training time from image resolution.

VII. Conclusion

In this work, we presented a novel approach for optimizing Visual Transformer architecture by replacing the computationally expensive self-attention layer with parameter-free multi-head token permutation heads. By leveraging random permutations and random sign flipping, we obtained an effective token mixing mechanism with linear $O(N)$ computational complexity, avoiding the quadratic cost of the standard self-attention. Moreover, we proposed a spectral spatial reduction technique using 2D Real FFT during the tokenization stage, which efficiently compresses input features while preserving global information.

Our experimental results on MNIST and CIFAR-100 datasets demonstrate that the proposed architecture maintains competitive accuracy compared to baseline ViT and F-Net models while significantly reducing model size and computational cost. Specifically, on the MNIST dataset, our method outperformed the standard ViT in accuracy (95.1% vs 93.0%) while reducing inference latency by approximately 60% and halving the parameter count. On the CIFAR-100 dataset, we achieved comparable validation performance with a 75% reduction in parameters compared to the baseline.

The deployment experiment evaluation on the ARM Cortex-A53 embedded platform proves the practical viability of our method for edge computing applications. The ability to run efficiently on CPU without relying on specialized matrix multiplication accelerators makes this approach highly accessible for low-power devices.

Acknowledgment

The research was supported by the Russian State Research grant FFZF-2025-0003.

References

[1] X. Wu, L. Jiang, P.-S. Wang, Z. Liu, X. Liu, Y. Qiao, W. Ouyang, T. He, and H. Zhao, "Point transformer v3: Simpler faster stronger," *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4840–4851, 2024.

[2] J. Lee-Thorp, J. Ainslie, I. Eckstein, and S. Onta ñón, "FNet: Mixing tokens with Fourier transforms," *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 4296–4313, Jul. 2022.

[3] Y. Zhang and X. Li, "Fast convolutional neural networks with fine-grained ffts," *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques*, p. 255–265, 2020. [Online]. Available: https://doi.org/10.1145/3410463.3414642

[4] E. Reis, T. Akilan, and M. Khalid, "Phasor-Driven Acceleration for FFT-based CNNs," 2024. [Online]. Available: http://arxiv.org/abs/2406.00290

[5] O. Rippel, J. Snoek, and R. Adams, "Spectral representations for convolutional neural networks," *Advances in Neural Information Processing Systems*, vol. 2015-January, pp. 2449–2457, 2015.

[6] J. Bruna and S. Mallat, "Invariant scattering convolution networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1872–1886, 2013.

[7] T. Ki and Y. Hur, "Deep scattering network with max-pooling," *2021 Data Compression Conference (DCC)*, pp. 348–348, 2021.

[8] F. Wolf-Monheim, "Spectral and Rhythm Features for Audio Classification with Deep Convolutional Neural Networks," 2024. [Online]. Available: http://arxiv.org/abs/2410.06927

[9] A. Kiruluta and S. Williams, "Reducing Deep Network Complexity via Sparse Hierarchical Fourier Interaction Networks," 2025. [Online]. Available: http://arxiv.org/abs/1801.01451

[10] J. Bai, F. Lu, K. Zhang *et al.*, *ONNX: Open Neural Network Exchange*, 2019, [Accessed March. 4, 2026]. [Online]. Available: https://github.com/onnx/onnx

[11] *Intel® Distribution of OpenVINO™ Toolkit — intel.com*, [Accessed 29-01-2026]. [Online]. Available: https://www.intel.com/content/www/us/en/developer/tools/openvino-toolkit/overview.html

[12] H. Jin, M. Xiao, Y. Yuan, X. Zhang, D. Yu, G. Zhang, and H. Wang, "DistrAttention: An Efficient and Flexible Self-Attention Mechanism on Modern GPUs," 2025. [Online]. Available: http://arxiv.org/abs/2507.17245

[13] N. Shazeer, "Fast transformer decoding: One write-head is all you need," 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:207880429

[14] J. Ainslie, J. Lee-Thorp, M. de Jong, Y. Zemlyanskiy, F. Lebron, and S. Sanghai, "GQA: Training generalized multi-query transformer models from multi-head checkpoints," *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 4895–4901, Dec. 2023. [Online]. Available: https://aclanthology.org/2023.emnlp-main.298/

[15] I. Beltagy, M. E. Peters, and A. Cohan, "Longformer: The Long-Document Transformer," 2020. [Online]. Available: http://arxiv.org/abs/2004.05150

[16] T. Dao, D. Y. Fu, S. Ermon, A. Rudra, and C. R é, "Flashattention: fast and memory-efficient exact attention with io-awareness," *Proceedings of the 36th International Conference on Neural Information Processing Systems*, p. 16, 2022.

[17] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. Gonzalez, H. Zhang, and I. Stoica, "Efficient memory management for large language model serving with pagedattention," p. 611–626, 2023. [Online]. Available: https://doi.org/10.1145/3600006.3613165

[18] N. Sevim, E. O. Ozyedek, F. Şahinuç, T. Ozates, and A. Koç, "Fast-fnet: Accelerating transformer encoder models via efficient fourier layers: N. sevim et al." *Signal, Image and Video Processing*, vol. 19, no. 12, p. 966, 2025.

[19] B. N. Patro, V. P. Namboodiri, and V. S. Agneeswaran, "Spectformer: Frequency and attention is what you need in a vision transformer," *2025 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pp. 9543–9554, 2025.

[20] J. Fein-Ashley, N. Gupta, R. Kannan, and V. Prasanna, "SPECTRE: An FFT-Based Efficient Drop-In Replacement to Self-Attention for Long Contexts." [Online]. Available: http://arxiv.org/abs/2502.18394

[21] "The FFT Strikes Back: An Efficient Alternative to Self-Attention," 2025. [Online]. Available: https://arxiv.org/pdf/2502.18394v1

[22] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," p. 6000–6010, 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[23] L. Deng, "The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web]," vol. 29, no. 6, pp. 141–142. [Online]. Available: https://ieeexplore.ieee.org/document/6296535

[24] *CIFAR-10 and CIFAR-100 Datasets*, [Accessed March. 4, 2026]. [Online]. Available: https://www.cs.toronto.edu/~kriz/cifar.html

[25] ONNX, *DFT operator in ONNX*, 2023, [Accessed March. 4, 2026]. [Online]. Available: https://onnx.ai/onnx/operators/onnx__DFT.html

[26] ONNX, *STFT operator in ONNX*, 2023, [Accessed March. 4, 2026]. [Online]. Available: https://onnx.ai/onnx/operators/onnx__STFT.html

[27] Intel, *Supported Operations in OpenVINO*, 2024, [Accessed March. 4, 2026]. [Online]. Available: https://docs.openvino.ai/2024/about-openvino/compatibility-and-support/supported-operations.html

[28] Google, *Supported Select TensorFlow operators for TensorFlow Lite*, 2025, [Accessed March. 4, 2026]. [Online]. Available: https://ai.google.dev/edge/litert/conversion/tensorflow/op_select_allowlist

[29] Google, *FourierTransformLayer in TensorFlow Models NLP*, 2024, [Accessed March. 4, 2026]. [Online]. Available: https://www.tensorflow.org/api_docs/python/tfm/nlp/layers/FourierTransformLayer

[30] NVIDIA Corporation, *NVIDIA TensorRT Operators Documentation*, 2024, [Accessed March. 4, 2026]. [Online]. Available: https://docs.nvidia.com/deeplearning/tensorrt/archives/tensorrt-861/operators/index.html

[31] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255. [Online]. Available: https://ieeexplore.ieee.org/document/5206848