Sparse Butterfly Matrix Attention for Enhancing Transformer Performance

Kuan-Chieh Wang, Jie Zhang, Bo-Wei Chiu, and Min-Te Sun

Department of Computer Science and Information Engineering
National Central University, Taoyuan, Taiwan 32001
blackpaladin00011@gmail.com, hazdzz@g.ncu.edu.tw, h23468270@g.ncu.edu.tw, msun@csie.ncu.edu.tw

ABSTRACT

To handle long sequential data, we propose Butterflyer, an efficient Transformer-based model. Butterflyer employs an innovative Butterfly-Attention mechanism with Butterfly matrices to enhance interaction pattern capture and reduce computation costs, as these Butterfly matrices preserve orthogonality or unitarity and have a sparse structure. We evaluate Butterflyer on the Long Range Arena (LRA) benchmark and find it outperforms state-of-the-art models in various tasks.

1. INTRODUCTION

Handling long sequential data is a significant challenge in AI, particularly for tasks like text and image classification. Attention mechanisms, such as those in the Transformer model [27], have transformed computer vision and NLP, but their quadratic complexity limits scalability for long sequences. Solutions like Linformer [29], Reformer [9], and Longformer [2] reduce complexity or combine local and global attention. However, further optimization is needed for real-time processing and extremely long sequences.

Studies have looked into improving Transformers by using low-rank and sparse factorization to handle data more efficiently. Paramixer [32] changes self-attention by removing dot products and softmax for added flexibility. The butterfly matrix, used in FFT, helps with performance through a structured approach [5]. Sapkota *et al.* [21] propose using the butterfly matrix in neural networks for better optimization of long sequences. However, combining these methods into one model could offer even greater benefits.

This paper presents Butterflyer, a model designed for efficiently handling long sequential data in LRA benchmark. Butterflyer enhances performance by employing Butterfly Matrices and novel attention mechanisms. Inspired by Paramixer [32], it replaces traditional softmax with Butterfly Matrices to handle high-rank attention efficiently. An MLP enhances the data. Although Butterflyer is very accurate, it may be slightly slower than Paramixer due to extra layer normalization.

In summary, the contributions of this research are:

- Developed Butterflyer for effective processing of long sequential data across various tasks.
- Integrated Butterfly Matrices into the attention mechanism for improved stability and accuracy.
- Implemented advanced preprocessing techniques to ensure high-quality data for accurate classification.
- Demonstrated that Butterflyer outperforms state-of-the-art models in LRA.

Inspired by Paramixer [32], which rethinks self-attention as a transformation block, eliminating dot-product and softmax operations for enhanced efficiency and flexibility, Butterflyer replaces traditional softmax in scaled dot-product attention with Butterfly Matrices. This approach maintains computational efficiency while capturing high-rank attention requirements. The preprocessed data is passed through a multilayer perceptron (MLP) to mix the input tensor columns, enhancing data representation. The main challenges are multi-task handling, efficient feature extraction , generalization and prevent from overfitting. Despite its superior accuracy, Butterflyer exhibits a little bit of slower processing speeds compared to Paramixer, possibly due to its additional layer normalization.

2. RELATED WORK

2.1 Attention mechanisms

Integrating attention mechanisms into neural networks has significantly advanced fields like computer vision and natural language processing (NLP). Xu *et al.* [31] introduce an attention-based model for image captioning, subsequently extending it to object detection and image generation. Wang *et al.* [28] develop an attention-based residual network for fine-grained image recognition, excelling at identifying subtle differences. Li [19] design an attention-based generative adversarial network (GAN)) for realistic image synthesis. In NLP, Bahdanau *et al.* [1] pioneer attention mechanisms for machine translation, setting a standard for sequence-to-sequence tasks. Luong *et al.* [14] enhance translation with global and local attention. Vaswani *et al.* [27] revolutionize NLP with the Transformer model's self-attention mechanism. Shen [22] introduce an adaptive attention span for machine translation and language modeling, achieving state-of-the-art results.

2.2 Matrix Decomposition and Sparse Factorization

Self-attention mechanisms use matrix factorization to break down input sequences into weighted combinations. Traditional low-rank matrix decomposition methods, like Singular Value Decomposition (SVD), decompose matrices into simpler, low-rank forms. SVD is often used in recommendation systems [10] to model user-item interactions. Nyström decomposition [30] reduces complexity in large-scale matrix operations, while CUR decomposition [16] creates low-rank approximations by selecting specific rows and columns, aiding dimensionality reduction. Advances like the Butterfly Transform (BFT) [26] and Butterfly MLP [21] enhance CNN efficiency and matrix operations.

Sparse factorization methods use sparse matrices with fewer non-zero elements for more efficient approximations. Non-negative Matrix Factorization (NMF) [11] is widely used in fields like bioinformatics and text mining. Chord Sparse Factorization [23] specifies non-zero positions using a modified Chord protocol, optimizing the matrix through non-parametric minimization. Parametric Sparse Factorization Attention [8] improves sparse matrices using the Adam algorithm.

3. PRELIMINARY

3.1 Transformer

Transformers use self-attention mechanisms, a key element in neural network architectures [27]. The Feed-Forward Network (FFN) is essential for processing features at each position within the Transformer model. Self-attention allows elements in a sequence to interact fully, crucial for computing representations, as shown in Equation 1.

$$\text{Attention}(Q, K, V) = AV, \tag{1}$$

where

$$A = \text{softmax}\left(\frac{QK^T}{\sqrt{D}}\right), \tag{2}$$

In this context, Q, K, and V represent the Query, Key, and Value matrices derived from the input sequence through linear transformations. The attention matrix A is calculated using the scaled dot product of Q and K. The Key dimensionality D scales the dot product, and the softmax function normalizes these values, applying weights to the Value matrices to compute the final weighted sum.

3.2 Sparse Attention with Paramixer

Paramixer reimagines self-attention in Transformers by eliminating dot-product and softmax, reducing computational bottlenecks and increasing flexibility [32]. It uses an MLP to mix tensor columns, efficiently capturing complex relationships. Stacked Paramixer blocks form ParamixerNet, which adapts to tasks using back-propagation and optimization algorithms like Adam. This approach improves performance and generalization in various NLP [15] and machine learning tasks [17].

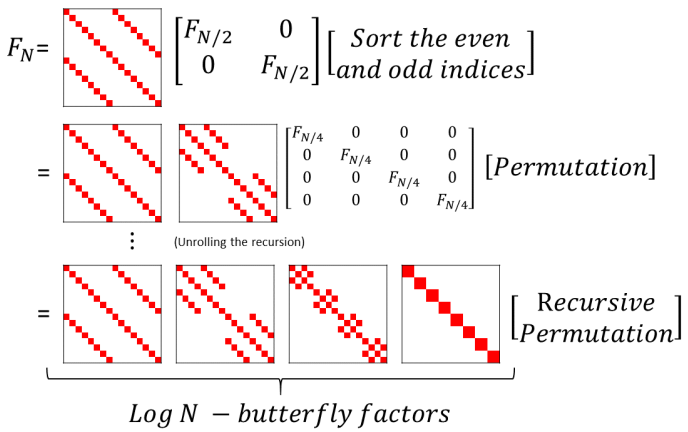

Figure 1: The Butterfly Matrix construction process. The red dots indicate non-zero elements.

3.3 Butterfly Matrix

The Butterfly Matrix, or Butterfly Factorization, is a structured matrix used in numerical analysis and signal processing, especially in FFT algorithms [5]. Its diagonal block design simplifies large linear transformations into smaller operations, leveraging data sparsity to reduce computational complexity. This enhances performance in tasks like spectral analysis, image processing, and data compression, as shown in Figure 1 [18].

The construction of a Butterfly Matrix typically involves the following steps:

1. **Initialization**: We start by initializing the Butterfly Matrix as an $N \times N$ identity matrix.
2. **Transformation Stage**: The matrix undergoes $\log_2(N)$ rounds of transformation. At each round s, the matrix is divided into 2^s blocks.
3. **Butterfly Operations**: Within each block, elements are mixed using specific patterns involving additions and multiplications, aiming to efficiently combine data and capture complex interactions.

Butterfly Matrices have been widely used in linear models but limited in nonlinear settings. Sapkota *et al.* [21] introduced the Non-Linear Butterfly Mixer, which integrates Butterfly Matrices into nonlinear operations, like MLPs and attention mechanisms. This approach enhances computational efficiency and parameter optimization, improving model performance in deep learning and expanding neural network design possibilities.

4. DESIGN

4.1 Problem Statement

Our goal is to develop a system that accurately classifies long sequential data (over 512 time steps) across diverse tasks. The system processes tokenized numerical arrays from various data types: images X_{img} converted into pixel sequences, text X_{text} into word or subword tokens,

pathfinder data X_{path} into coordinate sequences, and mathematical operations X_{math} into symbol sequences. The model extracts features, performs matrix operations, and applies classification techniques to ensure accurate results and good generalization across different data types.

4.2 Research Challenges

We plan to use advanced neural network architectures for our solution, but these face challenges when handling diverse tasks. Here are several considerations we have taken into account:

- **Multi-Task Handling:** The tasks in question are diverse, including but not limited to:
  1. Text Classification: Categorizing text documents based on content or context.
  2. Image Classification: Assigning images to predefined classes based on visual content.
  3. Pathfinder: Identifying paths between highlighted points in synthetic images.
  4. Mathematical Operations Analysis: Parsing and evaluating hierarchical sequences of mathematical operations.
  5. Document Retrieval: Evaluating a model's ability to encode and compare compressed representations of documents to determine similarity. This involves binary classification to identify citation links between documents.

  Each task requires specific adjustments to the neural network architecture to handle different sequence lengths and types, ensuring optimal performance across all tasks.
- **Efficient Feature Extraction:** Model performance hinges on extracting relevant features from input data. Traditional attention mechanisms may miss task-specific features. Enhancing feature extraction involves adjusting attention patterns or adding task-specific attention heads.
- **Generalization and Overfitting:** A key challenge is to ensure the model generalizes well across all tasks without overfitting any specific one.

Overcoming these challenges requires deep research into neural network architectures and innovative enhancements. Developing a model that efficiently handles multiple tasks while efficiently extracting task-specific features will advance in this field.

4.3 Proposed System Architecture

The proposed system architecture is shown in Figure 2. We preprocess input data types, such as images and text, converting them into numerical arrays. The data then undergoes Layer Normalization and Butterfly-Attention to extract relevant features. Each component will be detailed in the following sections.

- **Text Tokenization:** We create a list of unique characters from the input text, assign a token to each, and tokenize the text for further processing.

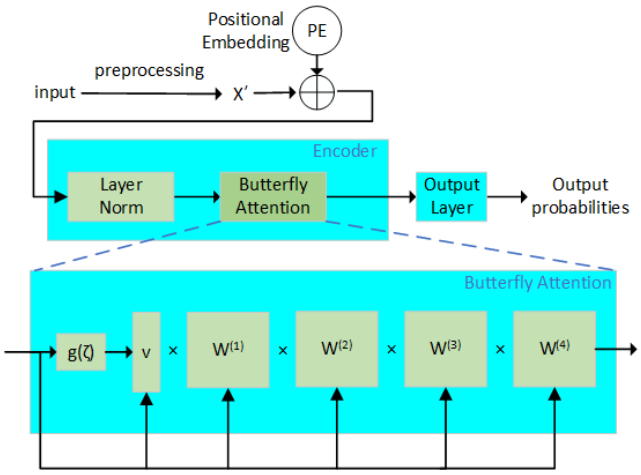

Figure 2: System Architecture

- **Image Transformation:** We convert each image to tensors, then use the squeeze operation to remove unnecessary dimensions and flatten them for model input.
- **Pathfinder Data Representation:**
  We convert the data into tensors, load the images, and create a pixel-to-index mapping. Each image is transformed into a sequence of pixel values, with target labels extracted. These preprocessed images and labels are used for subsequent model training and evaluation.
- **Mathematical Operations Encoding:** We read the sequences of the mathematical formula text, replace parentheses, and split the strings to extract tokens. A token-to-index mapping is created, and sequences are converted to index sequences with target labels extracted.
- **Document Retrieval Data Processing:** Document retrieval tasks involve two document sets: those to be retrieved and reference documents. We create a character dictionary, convert each document's text into integer sequences, concatenate sequences from both sets, and save the processed data for further use.

After the procedure, we apply padding and truncation to ensure uniform sequence lengths, typically 256, 512, or up to 4K. This standardization is essential for batch processing during model training, enabling effective feature extraction and classification.

The Butterflyer model processes various input types, such as images and text, as shown in Figure 2. We start by applying embedding and positional encoding to capture sequence information and dependencies. Layer normalization then stabilizes and speeds up training by ensuring uniform data scales, reducing overfitting, and addressing internal covariate shift. Inspired by Paramixer [32], we replace traditional softmax with the product of square matrices for scaled dot-product attention. Our Butterfly-Attention uses Butterfly Matrices [5] to apply efficient patterns and capture complex data relationships.

The preprocessed data is passed through an MLP $g : \mathbb{R}^d \to \mathbb{R}^d$ with weights ζ, mixing the input tensor columns

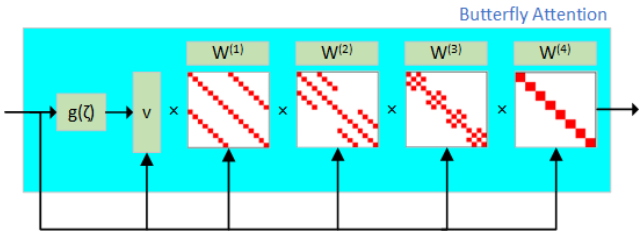

Figure 3: The Butterfly Attention architecture.

to create matrix V, enhancing feature representation. Next, we use butterfly decomposition to build butterfly matrices. Starting with an $N \times N$ identity matrix I and initializing B as I, we generate matrices $W^{(i)}$ through $\log_2(N)$ stages. Each stage captures different patterns, efficiently learning complex relationships, as shown in Figure 3.

The matrix V from the MLP is multiplied by the Butterfly Matrices $W^{(i)}$, enhancing data representation by combining the MLP's power with Butterfly Matrices' transformations.The processed data is then passed to a task-specific output layer, producing probabilities or predictions for classification, regression, or other objectives.

We use the Sophia optimizer [13], a gradient descent algorithm that dynamically adjusts learning rates for faster convergence and better performance, especially for large-scale neural networks.

Overall, Butterflyer leverages effective embedding, normalization, Butterfly-Attention, and the Sophia optimizer to deliver high-performance results for various predictive tasks.

5. EXPERMENT

This section evaluates Butterflyer against other state-of-the-art models [32, 12, 6]. We aim to show Butterflyer's effectiveness in capturing long sequence patterns and its superior performance across various tasks.

5.1 Experimental Environment

The experimental environment includes both hardware and software settings. The hardware setup consisted of an Intel Xeon Gold 6154 CPU @ 3.00GHz, a Tesla V100-SXM2 GPU, 60GB of RAM, and 30GB of shared memory. The software environment used a Linux distribution with kernel version 3.10.0-1127.el7.x86_64, Pytorch 12.1, and CUDA 12.3.

5.2 Dataset Information

We evaluated the Butterflyer model using the Long Range Arena (LRA) benchmark, designed to test how well efficient Transformer models manage long sequential data across various tasks.

- **Text Classification**: This task uses IMDb [15] reviews to test the model's ability to classify lengthy documents, focusing on sequences up to 4K characters and handling complex and unsegmented data.
- **Image Classification**: The CIFAR-10 dataset assesses the model's ability to 2D learn spatial relationships between pixels represented in a 1D sequence.
- **Pathfinder**: This task evaluates whether the model can determine if two points in a 32×32 image, treated as a 1024-length pixel sequence, are connected by a path.
- **ListOps (Hierarchical Data Parsing)**: This task tests the model's ability to reason over hierarchical sequences, using operators like MAX, MEAN, and SUM_MOD, within sequences up to 2K in length. For example, the sequence `[MAX 4 3 [MIN 2 3] 1 0 [MEDIAN 1 5 8 9 2]]` yields an output of 5. The model must predict one of ten output classes by understanding the hierarchical structure and operators in the input sequences.
- **Document Retrieval**: This task uses the ACL Anthology Network (AAN) dataset to test the model's ability to encode and compare compressed document representations, identifying citation links between papers. Each document is represented as a 4K byte/character sequence, making each comparison 8K in length.

5.3 Experimental Results and Analysis

Table 1 shows that Butterflyer performs strongly in LRA benchmark, particularly in text and image classification. Although Paramixer scored slightly higher in ListOps (39.71), Butterflyer was close with (39.51 ± 0.38), handling its capability in handling complex data. Butterflyer achieved the highest score in the retrieval task, proving its efficiency. While it performed slightly lower in the Pathfinder task, within 3% of the top score, it still outperformed other models like Paramixer and FNet in many areas. Butterflyer is effective for various tasks, making it a versatile choice for natural language processing, computer vision, and beyond.

Both models use sparse factorization to enhance efficiency. Paramixer optimizes non-zero positions in factor matrices, while Butterflyer employs a hierarchical butterfly matrix pattern, common in Fast Fourier Transform algorithms. Butterflyer's method yields higher accuracy but increases processing time for larger inputs.

Table 2 compares the average epoch time for Butterflyer and Paramixer on the LRA dataset. Butterflyer, while achieving higher accuracy, is slower than Paramixer, likely due to additional layer normalization. The speed difference is minor for most tasks but more noticeable in the retrieval task with larger input sizes (8K), suggesting that Butterflyer might need further optimization for large inputs.

5.4 Ablation Studies

This section analyzes the effect of Layer Normalization (LN) on the Butterflyer model. LN helps stabilize training and improves convergence. Testing with the LRA benchmark, results in Table 3 show that the butterfly matrix significantly enhances performance, especially in complex tasks like text classification and retrieval. While LN further improves accuracy and consistency, Butterflyer is still highly effective without it, particularly in image classification. This underscores the butterfly matrix's power and the added benefits of LN.

Table 1: The best result is in bold and the second best is underlined.

Model	ListOps	Text	Retrieval	Image	Pathfinder	Avg
Vanilla Transformer [27]	36.37	64.27	57.46	42.44	71.40	54.39
Sparse Transformer [3]	17.07	63.58	59.59	44.24	71.71	51.24
Reformer [9]	37.27	56.10	53.40	38.07	68.50	50.67
Longformer [2]	35.63	62.85	56.89	42.22	69.71	53.46
Linformer [29]	35.70	53.94	52.27	38.56	76.34	51.36
BigBird [33]	36.05	64.02	59.29	40.83	74.87	55.01
Linear Transformer [7]	16.13	65.90	53.09	42.34	75.30	50.55
Sinkhorn Transformerr [25]	33.67	61.20	53.83	41.23	67.45	51.29
Performerr [4]	18.01	65.40	53.82	42.77	77.05	51.41
Synthesizer [24]	36.99	61.68	54.67	41.61	69.45	52.88
cosFormerr [20]	37.90	63.41	61.36	43.17	70.33	55.23
Flowformer [6]	38.70	64.29	<u>62.24</u>	43.20	73.95	56.48
FNet [12]	35.33	65.11	59.61	38.67	<u>77.80</u>	55.30
Paramixer [32]	**39.71**	<u>78.87</u>	52.12	<u>44.68</u>	**79.16**	<u>58.91</u>
Butterflyer	<u>39.51 ± 0.38</u>	**81.71 ± 0.36**	**68.37 ± 0.44**	**48.84 ± 1.31**	76.79 ± 0.67	**63.04**

Table 2: Processing Times per Epoch for Each Task (in seconds)

Task	Paramixer	Butterflyer
Image	52s	70s
Text	205s	223s
Pathfinder	1139s	1170s
ListOps	1269s	1307s
Retrieval	923s	1704s

6. CONCLUSION

This paper introduces Butterflyer, a model designed to efficiently process long sequential data across various tasks like text and image classification, pathfinding, mathematical operations, and document retrieval. Butterflyer uses Butterfly Matrices and novel attention mechanisms to overcome limitations of traditional self-attention, and employs layer normalization and the Sophia optimizer to enhance training efficiency. Experimental results show that Butterflyer outperforms state-of-the-art models in accuracy but has slower processing speeds, particularly in tasks with large input sizes. Like many Transformer variants, Butterflyer struggles with long sequences. For instance, most transformer-like models do not succeed in the Pathfinder-X task in the LRA. Future work could focus on optimizing processing speed, improving spatial dependency handling, and exploring broader applications.

REFERENCES

[1] Bahdanau, D.; Cho, K.; and Bengio, Y. 2014. Neural Machine Translation by Jointly Learning to Align and Translate. *arXiv preprint arXiv:1409.0473*.

[2] Beltagy, I.; Peters, M. E.; and Cohan, A. 2020. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*.

[3] Child, R.; Gray, S.; Radford, A.; and Sutskever, I. 2019. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*.

[4] Choromanski, K.; Likhosherstov, V.; Dohan, D.; Song, X.; Gane, A.; and Sarlos, T. 2020. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*.

[5] Dao, T.; Gu, A.; Eichhorn, M.; Rudra, A.; and Ré, C. 2019. Learning fast algorithms for linear transforms using butterfly factorizations. In *International conference on machine learning*, 1517–1527. PMLR.

[6] Huang, Z.; Shi, X.; Zhang, C.; Wang, Q.; Cheung, K. C.; Qin, H.; Dai, J.; and Li, H. 2022. Flowformer: A Transformer Architecture for Optical Flow. In *European Conference on Computer Vision*, 668–685. Springer Nature Switzerland.

[7] Katharopoulos, A.; Vyas, A.; Pappas, N.; and Fleuret, F. 2020. Transformers are RNNs: Fast autoregressive transformers with linear attention. In *International Conference on Machine Learning*, 5156–5165. PMLR.

[8] Khalitov, R.; Yu, T.; Cheng, L.; and Yang, Z. 2022. Sparse factorization of square matrices with application to neural attention modeling. *Neural Networks*, 152: 160–168.

[9] Kitaev, N.; Kaiser, Ł.; and Levskaya, A. 2020. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*.

[10] Koren, Y.; Bell, R.; and Volinsky, C. 2009. Matrix factorization techniques for recommender systems. *Computer*, 42(8): 30–37.

[11] Lee, D.; and Seung, H. S. 2001. Algorithms for non-negative matrix factorization. In *Advances in neural information processing systems*, volume 13, 556–562.

Table 3: Evaluation result of ablation studies.

Model	ListOps	Text	Retrieval	Image	Pathfinder	Avg
Paramixer [32]	39.71	78.87	52.12	44.68	79.16	58.91
Butterflyer without LN	38.82 ± 0.43	79.91 ± 0.51	68.31 ± 0.27	49.31 ± 1.47	77.37 ± 1.22	62.74
Butterflyer	39.51 ± 0.38	81.71 ± 0.36	68.37 ± 0.44	48.84 ± 1.31	76.79 ± 0.67	63.04

[12] Lee-Thorp, J.; Ainslie, J.; Eckstein, I.; and Ontanon, S. 2021. Fnet: Mixing tokens with fourier transforms. *arXiv preprint arXiv:2105.03824*.

[13] Liu, H.; Li, Z.; Hall, D.; Liang, P.; and Ma, T. 2023. Sophia: A scalable stochastic second-order optimizer for language model pre-training. *arXiv preprint arXiv:2305.14342*.

[14] Luong, M.-T.; Pham, H.; and Manning, C. D. 2015. Effective Approaches to Attention-based Neural Machine Translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*.

[15] Maas, A.; Daly, R. E.; Pham, P. T.; Huang, D.; Ng, A. Y.; and Potts, C. 2011. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*, 142–150.

[16] Mahoney, M. W.; and Drineas, P. 2009. CUR matrix decompositions for improved data analysis. *Proceedings of the National Academy of Sciences*, 106(3): 697–702.

[17] Nangia, N.; and Bowman, S. R. 2018. Listops: A diagnostic dataset for latent tree learning. *arXiv preprint arXiv:1804.06028*.

[18] Prabhu, A.; Farhadi, A.; and Rastegari, M. e. a. 2020. Butterfly transform: An efficient FFT-based neural architecture design. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 12024–12033.

[19] Qian, R.; Tan, R. T.; Yang, W.; Su, J.; and Liu, J. 2018. Attentive Generative Adversarial Network for Raindrop Removal from A Single Image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

[20] Qin, Z.; Sun, W.; Deng, H.; Li, D.; Wei, Y.; Lv, B.; Yan, J.; Kong, L.; and Zhong, Y. 2022. Cosformer: Rethinking softmax in attention. *arXiv preprint arXiv:2202.08791*.

[21] Sapkota, S.; and Bhattarai, B. 2023. Dimension Mixer: A Generalized Method for Structured Sparsity in Deep Neural Networks. *arXiv preprint arXiv:2311.18735*.

[22] Shen, Z.; Zhang, M.; Zhao, H.; Yi, S.; and Li, H. 2021. Efficient Attention: Attention With Linear Complexities. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, 3531–3539.

[23] Stoica, I.; Morris, R.; Karger, D.; Kaashoek, M. F.; and Balakrishnan, H. 2001. Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM computer communication review*, 31(4): 149–160.

[24] Tay, Y.; Bahri, D.; Metzler, D.; Juan, D.-C.; Zhao, Z.; and Zheng, C. 2021. Synthesizer: Rethinking self-attention for transformer models. In *International conference on machine learning*, 10183–10192. PMLR.

[25] Tay, Y.; Bahri, D.; Yang, L.; Metzler, D.; and Juan, D.-C. 2020. Sparse sinkhorn attention. In *International Conference on Machine Learning*, 9438–9447. PMLR.

[26] Vahid, K. A.; and et al. 2020. Butterfly transform: An efficient fft based neural architecture design. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE.

[27] Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; and Polosukhin, I. 2017. Attention Is All You Need. In *Advances in Neural Information Processing Systems*.

[28] Wang, F.; Jiang, M.; Qian, C.; Yang, S.; Li, C.; Zhang, H.; Wang, X.; and Tang, X. 2017. Residual Attention Network for Image Classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

[29] Wang, S.; Li, B. Z.; Khabsa, M.; Fang, H.; and Ma, H. 2020. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*.

[30] Williams, C. K. I.; and Seeger, M. 2001. Using the Nystrom method to speed up kernel machines. In *Advances in neural information processing systems*, 682–688.

[31] Xu, K.; Ba, J.; Kiros, R.; Cho, K.; Courville, A.; Salakhudinov, R.; Zemel, R.; and Bengio, Y. 2015. Show, Attend and Tell: Neural Image Caption Generation with Visual Attention. In *Proceedings of the 32nd International Conference on Machine Learning*.

[32] Yu, T.; Khalitov, R.; Cheng, L.; and Yang, Z. 2022. Paramixer: Parameterizing mixing links in sparse factors works better than dot-product self-attention. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 691–700.

[33] Zaheer, M.; Guruganesh, G.; Dubey, K. A.; Ainslie, J.; Alberti, C.; Ontanon, S.; Pham, P.; Ravula, A.; Wang, Q.; Yang, L.; and Ahmed, A. 2020. Big Bird: Transformers for longer sequences. *Advances in Neural Information Processing Systems*, 33: 17283–17297.