Machine Learning (2025) 114:139
https://doi.org/10.1007/s10994-025-06768-3

Drop-in efficient self-attention approximation method

Damien François[1] · **Mathis Saillot**[3] · **Jacques Klein**[1] · **Tegawendé F. Bissyandé**[1] · **Alexander Skupin**[2,4,5]

Received: 8 February 2025 / Revised: 10 March 2025 / Accepted: 16 March 2025

Abstract
Transformers have achieved state-of-the-art performance in most common tasks to which they have been applied. Those achievements are attributed to the Self-Attention mechanism at their core. Self-Attention is understood to map the relationship between tokens of any given sequence. This exhaustive mapping incurs massive costs in memory and inference time, as Self-Attention scales quadratically with regard to sequence length. Standard Self-Attention has required increasingly large compute and memory usage when applied to long input sequences because of this memory and time bottleneck. Efficient Transformers emerged as performant alternatives demonstrating good scalability and occasionally better tracking of long-range dependencies. Their efficiency gains are obtained through different methods, usually focusing on the linear scaling of the attention matrix through sparsification, approximation, or other methods. Among existing approaches, those using low-rank approximation present particular advantages because of their compatibility with standard Self-Attention-based models, allowing for weight transfers and other time-saving schemes. More recently, hardware-aware versions of Self-Attention (e.g., FlashAttention) have mitigated all memory bottlenecks and have alleviated its compute burden. Unfortunately, hardware-aware Self-Attentions have stricter hardware compatibility requirements making Efficient Transformers still relevant for use on older or less powerful hardware. Furthermore, some Efficient Transformers can even be applied in an hardware-aware manner to further improve training and inference speed. In this paper, we propose a novel linear approximation method for Self-Attention inspired by the CUR approximation method. This method, proposed in two versions (one leveraging FlashAttention), is conceived as a drop-in replacement for standard Self-Attention with weights compatibility. Our method compares favorably to standard Transformers' and Efficient Transformers' performances on varied tasks and demonstrates a significant decrease in memory footprint as well as competitive performance in training speed, even compared to similar methods.

Keywords Deep learning · Machine learning · Transformers · Self-attention · Attention approximation

Editors: Concha Bielza, Ana Carolina Lorena, Tiago A. de Almeida.

Damien François, Mathis Saillot, Jacques Klein, Tegawendé Bissyandé and Alexander Skupin have contributed equally.

Extended author information available on the last page of the article

1 Introduction

Transformers (Vaswani et al. 2017) have continually demonstrated in recent years their ability to achieve state-of-the-art (SOTA) performance in most tasks to which they are applied. They have, in all aspects, supplanted their predecessor, the Recurrent Neural Networks (RNN). In contrast with the sequential nature of the RNNs, the fundamental building block of any Transformer, the Self-Attention mechanism, is based on highly parallelizable computations. Self-Attention is considered a mapping of the relationship between tokens of an input sequence. The weak inductive bias of Transformers makes them uniquely agnostic to input types if an adequate tokenization procedure is applied to the original input (Dosovitskiy et al. 2020). It is, however, a double-edged sword that makes them perform poorly in data-starved scenarios, and often makes unsupervised or self-supervised pretraining an essential step to attain SOTA performance.

Despite their many advantages, one main drawback can be found in the very nature of Self-Attention as an exhaustive mapping of token relationships. An increasing amount of tasks demand longer and longer context. This increase of the context for Transformers is achieved by simply increasing the size of the input sequence. For any input sequence of length n, the attention matrix computed during inference is of size $n \times n$. Attention, therefore, scales quadratically with regard to input length, incurring a memory and time complexity of $O(n^2)$, which is a problem for long sequences. This quadratic complexity entails that most of the cost in time and memory of Transformer models when used in real-world tasks are attributable to the Attention matrix and not the trainable weights.

To alleviate this obvious performance bottleneck, the field of **Efficient Transformers (ET)** has emerged. All ETs endeavor to linearize the cost of Attention by computing only part of it, approximating it, or reworking its underlying mechanisms.

We differentiate 3 main groups of methods: Fixed pattern methods, learned pattern methods, and approximation methods. In the following work, we focus on the latter method for its overall performance, its ability to be seamlessly integrated into existing models, and the potential compatibility of the trained weights with standard Self-Attention Transformers.

More recently, FlashAttention (Dao et al. 2022; Dao 2023) has solved the memory footprint problems of Transformers and has introduced huge increases in training and inference speeds. Those methods suffer from the way they leverage specific hardware features by limiting their compatibility to more recent GPU architectures.

In this paper, we propose two implementations of a Self-Attention approximation method of $O(n)$ complexity in time and memory. One implementation leverages FlashAttention and the recent advances in GPU architecture, and another better fits the constraints of old GPU architecture. The method computes only a small fraction of the original Attention matrix through the selective multiplication of tensors in the Keys and Queries tensors. Although only part of the Attention matrix is computed, the method endeavors to recreate missing data through the use of a method inspired by the CUR matrix approximation method (Drineas et al. 2007), which uses a subset of columns, rows and the Moore-Penrose pseudo-inverse of the intersecting matrix to approximate a large matrix. We propose a new architecture, named CURformer architecture to achieve competitive or superior results compared to similar methods on various tasks that include input types of varied lengths, such as in the Long Range Arena (LRA) benchmark (Tay et al. 2020). We observe also good potential from this approach for fine-tuning from existing pre-trained weights and direct deployment without further training. We conceived this approach as a method

readily interoperable with standard Self-Attention, which aims to increase efficiency in speed and memory compared to standard attention while demonstrating better long-range dependency tracking in long sequences.

The contributions of our work are as follows:

- We propose an efficient Self-Attention mechanism. The approach, inspired by the CUR approximation method, demonstrates a notable decrease in memory footprint and in latency.
- We implement a hardware-aware version of our method which is faster than Flash-Attention 2
- We empirically demonstrate that (1) our proposed approach yields competitive results to similar approaches on varied tasks; (2) our method is efficient in the context of fine-tuning from standard Self-Attention models and their pre-existing trained weights.
- We show that our Self-Attention approximation method demonstrates some trained weights compatibility without retraining or fine-tuning especially when paired with exponential moving average augmented Self-Attention.

2 Related work

Efficient Transformers have been an active field of research, their aim is to alleviate the main bottlenecks of Transformers. We will quickly explore related work in the domain of ETs.

Sparsification of Attention is a common approach that limits the tokens over which Attention operates. Methods with simple fixed patterns such as Blockwise Attention (Qiu et al. 2019), Local Attention (Parmar et al. 2018), Strided Patterns Attention (Beltagy et al. 2020), as well as a combination of fixed pattern methods such as Axial Attention (Ho et al. 2019), Sparse Attention (Child et al. 2019) are good illustrations of this approach.

Extensions of the aforementioned approach are the **learnable patterns** methods that introduce token relevance for the purpose of assigning tokens to buckets or clusters, the Routing Transformer (Roy et al. 2021) uses k-mean for clustering, Reformer (Kitaev et al. 2020) a hash-based token similarity score to achieve similar clustering dynamics and learn patterns. Unexpectedly a reintroduction of **recurrence** from their defunct predecessor is used in some ETs to connect blocks (Dai et al. 2019).

Finally, the domain of **low-rank and kernelized** methods that uses approximation methods pre-Attention or on Attention itself. Using the kernel trick, the Performer model (Choromanski et al. 2020) avoids computing the $n \times n$ Attention matrix. Linear Transformer (Katharopoulos et al. 2020) and Random Feature Attention (Peng et al. 2021) use similar kernelization methods to accomplish their linearization of Attention. The Nyströmformer (Xiong et al. 2021) is based on the Nyström approximation method. Through what can be viewed as a mean pooling operation along the Queries and Keys vectors, it creates landmark vectors of lower dimensions. With a fixed number of landmarks, Attention complexity increases linearly with sequence size. Dynamic Bilinear Low-Rank Attention (Qin et al. 2022) employs approximation methods pre-Attention to compress the projected Queries and Keys vectors which are used in standard Attention to generate the full-scale Attention matrix, hence achieving a reduction in Attention size. The Singularformer (Wu et al. 2023) model trains small neural networks to learn the singular value decomposition of the Attention matrix to create a linear Self-Attention method.

Beyond Attention adaptations, work has been done on other facets of Transformers to increase efficiency. The Mega (Ma et al. 2022) model adds biases in the Attention matrix and uses exponential moving average (EMA) on the input to propagate context and add bias to the input sequence, which allows for great accuracy in benchmarks with fewer Attention heads. In their MegaChunk version the EMA is used to bridge the context between chunk of the input sequence allowing for a decrease in the size of Attention.

Methods using rotary positional encoding like the YaRN (Peng et al. 2023) method are used to train models to generalize beyond the sequence length in their pre-training. This allows for training on shorter sequences and for a smaller amount of training steps. Finally methods like LoRA (Hu et al. 2021) focus on the low-rank adaptation of trainable weights in the model instead of attention. In the context of fine-tuning it lessens the training time and the amount of trainable weights to update.

It is important to note that we do not consider methods that do not modify Self-Attention to be competing methods, because their approach can be combined with ours, like our tests with MEGA will attest in later sections.

Our approach fits squarely within the low-rank category, with a particular emphasis on pushing the efficiency of the model and exploring weight compatibility.

3 Background

3.1 Self-attention

Self-Attention is usually understood as a mapping of the learned relationships between tokens of a sequence. More formally, the input sequence is projected into embedding spaces through learnable weights forming the Keys (K), Queries (Q), and Values (V) matrices.

Attention can be defined as follows:

$$\underbrace{softmax\left[\frac{QW^Q(KW^K)^T}{\sqrt{d_k}}\right]}_{A} \tag{1}$$

Integrated into the full Self-Attention mechanism as illustrated in the following:

$$SelfAttention(QW^Q, KW^K, VW^V) = AVW^V, \tag{2}$$

with W^Q, W^K, W^V the weight matrices where $W^Q, W^K, W^V \in \mathbb{R}^{n\times d}$, n the sequence length and d the embedding dimension. As defined in Eq. 2, Attention includes a mapping of similarity score between each projected token in Q and K with each other, in the form of $A \in \mathbb{R}^{n\times n}$. This exhaustive context mapping requires the multiplication of Q and K of size $n \times d$, which is a process of $O(n^2)$ complexity in time and memory.

3.2 CUR approximation method

As represented in Fig. 1, the CUR method Drineas et al. (2007) approximates a low rank matrix A through the creation of three matrices C, U^+, and R. They are composed of a subset of columns (C), as well as a subset of rows (R) from the matrix A, and the

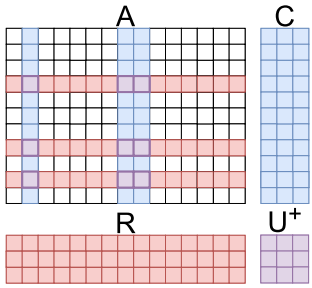

Fig. 1 CUR approximation method as traditionally applied to a big matrix to reduce memory utilization but keeping interpretability and the underlying structure of the original data. With columns in C, rows in R and U^+ the pseudoinverse of U the intersection of C and R

Moore-Penrose inverse Penrose (1955), (also called pseudoinverse) of the matrix U (U^+) defined as the intersection of C and R. By computing the product of the C, U^+, and R matrices, we can then reconstruct an approximation of the original matrix.

The method is often used as a dimensionality reduction method that captures parts of the original data. It also aims to preserve the structure of the data by selecting columns and rows with high statistical significance. Finally, because all elements in the CUR method are sampled from real elements of the matrix to approximate, findings or metrics computed on these elements can relate directly to the original data from the matrix, allowing for great interpretability of any results.

Not all those advantages are directly transferable to our method. Due to the absence in memory of the matrix to approximate, because the goal is not to compute a matrix of $O(n^2)$ complexity, the statistical significance of columns and rows is unknown before selection. The row-wise application of the Softmax activation functions also obfuscates the direct correlation between the columns of the original matrix and the C matrix in our approach.

In this sense our method is more so inspired by the CUR method as opposed to a direct application of the method to Self-Attention.

4 Attention approximation

As illustrated in Fig. 2, through the multiplication of Q and $\hat{K}^T$ we can obtain a matrix composed of a subset of selected columns from matrix QK^T. Similarly, by multiplying $\hat{Q}$ to K^T we obtain a matrix composed of a subset of selected rows from matrix QK^T. We obtain C and R by applying a row-wise Softmax activation function to both previously mentioned matrices. U is composed of a subset of rows of C. We obtain RV through the matrix multiplication of R and V. U^+ is obtained through the application of an iterative Moore-Penrose inverse algorithm to tensor U. To obtain the output we simply compute the following matrix multiplication: CU^+RV. In all implementations, RV is computed and available throughout the CUR process. To get the closest output to standard Self-Attention, and because RV contains a subset of rows of AV, we can put rows of RV at the corresponding indices in the output. This can allow for a marginal decrease in the overall approximation error between the output from our method and Self-Attention at a negligible cost.

4.1 Softmax scaling

Before the use of the Softmax activation function, the CUR method could be applied without compromising the approximation of QK^T. However, the independent

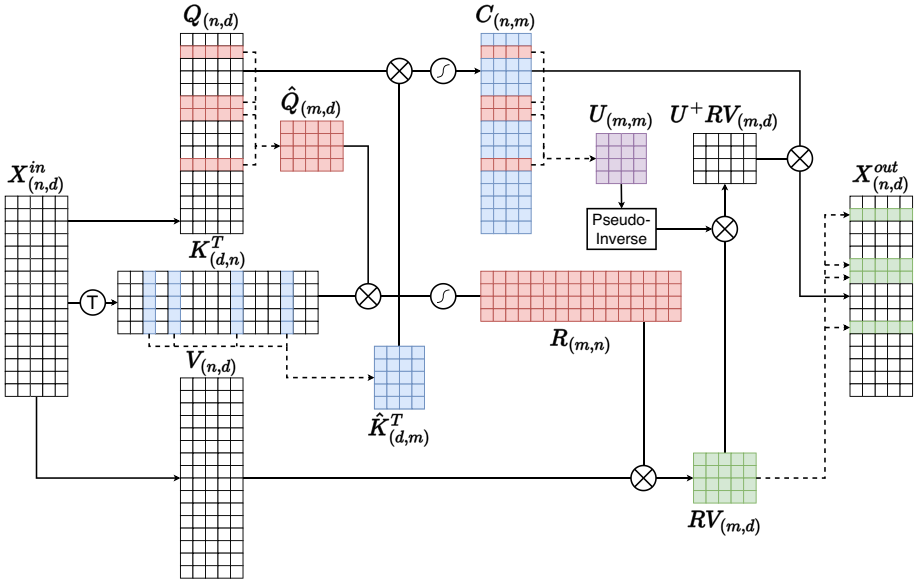

Fig. 2 CUR inspired method for Self-Attention linearization. Selected rows from Q are represented in red and selected columns from K are in blue. The green rows of X^{out} are the corresponding rows of RV (Color figure online)

application of the Softmax activation function to $Q\hat{K}^T$ and $\hat{Q}K^T$, respectively subsets of columns and rows of QK^T, is an obstacle to a good approximation. In standard Self-Attention, the Softmax activation function is applied row-wise to QK^T to obtain the final Attention matrix A as shown in Eq. 3.

$$A_{ij} = \frac{\exp\left(\left[QK^T\right]_{ij}\right)}{\sum_{k=1}^{n} \exp\left(\left[QK^T\right]_{ik}\right)} \qquad i,j \in 1,2,...,n \tag{3}$$

In the case of $\hat{Q}K^T$ being rows of QK^T, after the application of Softmax, the resulting matrix R is an exact replica of rows of A. However, with C, the row-wise application of Softmax creates columns that cannot be exact replicas of the corresponding columns of A. As described in Eq. 3 the Softmax function normalizes all values in the row of the affected matrix. All resulting values are between 0 and 1 and their row-wise sum equals 1. Applied to rows of differing lengths in C and A it introduces a scaling mismatch. However, we find through experimentation (cf. Section 6), that it is an acceptable inaccuracy for the purpose of approximating A using our method.

The same problem arises with U. If created from the intersection of $Q\hat{K}^T$ and $\hat{Q}K^T$, the row-wise application of Softmax would result in a similar scaling issue as in C. To get an adequate U, truly replicating elements of A, U could be created from elements of R with correct corresponding values. However, we experimentally verified that the only way for our method to generate adequate approximations of A is to maintain similar scaling in C and U in the axis to which the Softmax activation function is applied.

Hence, as illustrated in Fig. 2, U is created from row selections on the softmaxed C tensor.

4.2 Complexity analysis

For the complexity analysis of our Self-Attention approximation method (henceforth identified in tables, graphs, and figures with the prefix or suffix CUR), we need to compute it for the column and row selection, the computation of C, U, and R, the Moore-Penrose inverse, and matrix multiplications. Computations and selections all start, as in standard Attention, from the Queries, Keys and Values tensors all of them of size $n \times d$ with n the input sequence length and d the per-Attention head embedding size. Selecting m arbitrary rows and columns in the Queries and Keys tensors, incurs a time complexity of $O(md)$. C and R share a similar time complexity of $O(nmd)$ for the matrix multiplication and $O(nm)$ for the Softmax operation, giving a final complexity of $O(nmd)$. U can be extracted from C in $O(md)$. Following the pseudoinverse calculation in Xiong et al. (2021), the reported complexity is $O(m^3)$. The matrix multiplication for RV is in $O(mnd)$, and U^+RV is computed in $O(m^2d)$. The final matrix multiplication has a complexity of $O(mnd)$ and copying the rows of RV into the output is in $O(md)$. Using each individual complexity analysis gives a total of $O(3md + 4nmd + m^2d + m^3)$ which reduces to $O(nmd + m^2d + m^3)$ in time complexity. In terms of memory, we need to store the matrices $\hat{Q}$, $\hat{K}$, C, U, R, RV, U^+RV as well as the index list for a total complexity of $O(md + md + mn + m^2 + nm + md + md + m)$ which is $O(nm + m^2 + md)$. We can observe that when we choose a fixed number of selected columns and rows m such that $m << n$ and that by convention Dao (2023), $d << n$, the complexity of our method is now $O(n)$. In the following section we endeavour to empirically confirm this complexity analysis by using two different implementations of our method in ideal conditions stripped of extraneous factors.

5 Implementation

5.1 Attention approximation with Pytorch

To explore the approach we start with a Pytorch version to maintain compatibility with legacy hardware.

The implementation described in Algorithm 1 creates the $\hat{Q}$ and $\hat{K}$ matrices as separate from Q and K, as depicted in Fig. 2, despite the fact that they are simply a subset of rows and columns. Hence copying redundant data in memory to be able to compute C and R. At the end of the forward pass, $\hat{Q}$, $\hat{K}$, C, R, U, U^+, RV, U^+RV, and the output are all resident in memory simultaneously.

The main advantage of this implementation compared to competing methods is that it uses a subset of already existing matrix data used for standard Attention without any need for additional computations.

Algorithm 1 CUR Attention

function CURATTENTION($Q, K, V, k\_indices, q\_indices, n\_iter$)
  $\hat{Q} \leftarrow \text{index\_select}(Q, q\_indices)$
  $\hat{K} \leftarrow \text{index\_select}(K, k\_indices)$
  $C \leftarrow \text{softmax}(Q \times \hat{K}^T)$
  $R \leftarrow \text{softmax}(\hat{Q} \times K^T)$
  $U \leftarrow \text{index\_select}(C, q\_indices)$
  $U^+ \leftarrow \text{pinv}(U, n\_iter)$
  return $C(U^+(R \times V))$
end function

5.2 FlashAttention approximation with Triton

We investigate the applicability of our method to the hardware-aware paradigm. To do this, we implemented a version of approximated FlashAttention with Triton (Tillet et al. 2019) to determine whether the same performance gains can be achieved as with the Pytorch implementation. Our custom FlashAttention is composed of three custom Triton kernels and makes use of the same iterative pseudo-inverse algorithm used in our Pytorch implementation.

Hardware-aware Attention, known as FlashAttention (Dao et al. 2022) and its follow-up version FlashAttention 2 (Dao 2023), use custom kernels to compute A and its Softmax, as well as its matrix multiplication with V in the SRAM. It endeavors to avoid the need to store any Self-Attention data in VRAM and to fuse the Softmax operation to the matrix multiplication operations in SRAM, reducing memory hit rates and dramatically decreasing its overall computation cost. We take inspiration from the kernels found in FlashAttention 2 for our implementation.

As shown in Algorithm 2, there is no need to store $\hat{Q}$, $\hat{K}$, C and R in VRAM. Hence we have only RV, U, U^+, U^+RV, and the output resident in memory at the end of the forward pass. Each kernel in Algorithm 2 fuse matrix multiplication and softmax kernels to increase computation speed and decrease memory footprint in the VRAM.

Algorithm 2 CUR FlashAttention

function CURFLASHATTENTION($Q, K, V, k\_indices, q\_indices, n\_iter$)
  $RV \leftarrow \text{fused\_kernel\_rv}(Q, K, V, q\_indices)$
  $U \leftarrow \text{fused\_kernel\_u}(Q, K, k\_indices, q\_indices)$
  $U^+ \leftarrow \text{pinv}(U, n\_iter)$
  $U^+RV \leftarrow U^+ \times RV$
  return $\text{fused\_kernel\_out}(Q, K, V, U^+RV, k\_indices)$
end function

As represented in Fig. 3 the first kernel computes RV containing selected rows from AV. The second kernel computes U with the correct scaling of values mentioned in Sect. 4, and outputs U to the VRAM. For the iterative pseudo-inverse resulting in U^+ and its

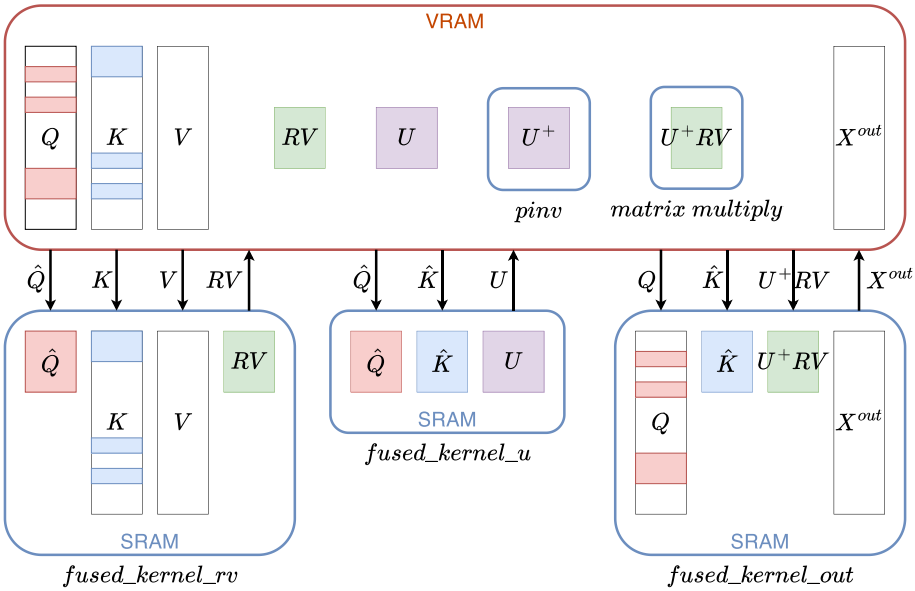

Fig. 3 Details of our approximated FlashAttention implementation with memory transfers visualization

multiplication to RV, no fused computations are necessary, hence we use the default matrix multiply functions of Pytorch to obtain U^+RV. The last kernel computes C and its matrix multiplication with U^+RV.

5.3 Self-attention approximation Vs standard self-attention (latency and memory)

We now empirically compare the latency and the memory footprint of our Pytorch and hardware-aware methods to Self-Attention and FlashAttention. To that end, we benchmark all of them in ideal circumstances. All extraneous layers, normalizations, or processes otherwise unrelated to Self-Attention have been stripped to measure the difference in speed and memory consumption between each method with minimal interference of other parameters.

Experimental Protocol: We instantiate a Transformer block with 16 Attention heads and a dimension per head of 64. To benchmark the speed and memory consumption of both methods, we create random tensors as input in batches of 64 sequences. We use sequences of 1024, 2048, 3072, and 4096 elements. For our method, we select 32, 64, and 128 indices from Q and K, with 6 iterations to compute the pseudoinverse.

The speed benchmark loops 100 times as a warm-up and executes both standard Attention and our approximated Attention 1000 times. Average latency and the average allocated memory during processing in Fig. 4.

Results: The results in Fig. 4 illustrate the quadratic cost in latency and memory caused by the increase in sequence size with Self-Attention. Quadrupling the sequences' size causes an increase of an order of magnitude in the latency and memory footprint of Self-Attention. In contrast, our method scales linearly with regard to sequence size, in both latency and memory, confirming the complexity analysis in section 4.2.

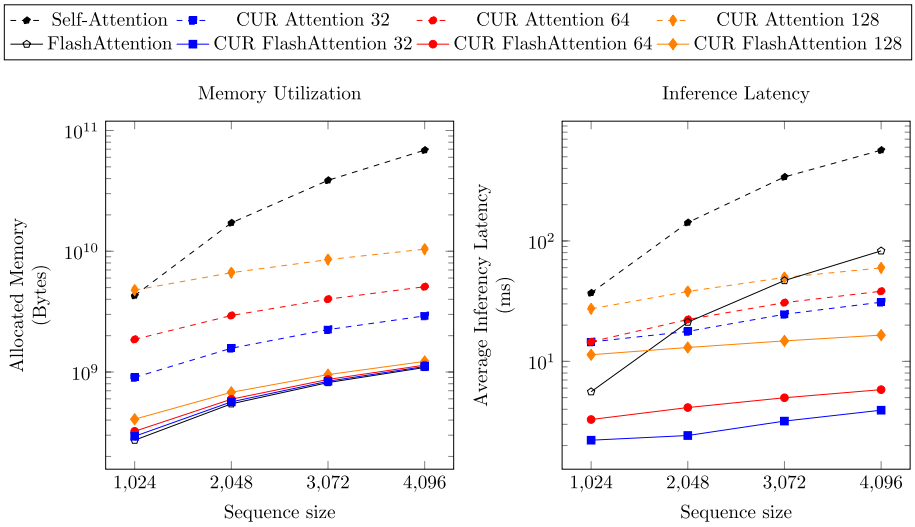

Fig. 4 Average Latency in ms (in log scale) and Memory allocated in Bytes (in log scale) for FlashAttention, Self-Attention and their approximated versions with a selection of 32, 64, or 128 indices depending on sequence size, for a batch of 64

The results in Fig. 4 also indicate that, though not as pronounced as with the Pytorch implementation, the latency difference between FlashAttention and our method is still present. Both follow their expected behavior; FlashAttention while faster than standard Self-Attention still suffers from a quadratic increase in inference time when the length of the sequence increases. Our method also follows the same linear increase as the Pytorch implementation when sequences' lengths increase. We can observe that our method is only slower than FlashAttention when 128 indices are selected on the sequence of length 1024. Finally, we can see that memory consumption between FlashAttention and the FlashAttention approximation method is of the same order of magnitude. Our version, as seen in Fig. 3, stores more tensors in VRAM. However, their size increases linearly with the input sequence size hence the approximated FlashAttention method, although slightly heavier in memory consumption, keeps the linear characteristics of FlashAttention.

6 LRA benchmark

Finally, we train multiple versions of our method on LRA to evaluate its performance compared to various ETs, as well as to evaluate the performance differences between every selection method when trained from scratch. The LRA benchmark (Tay et al. 2020) has been conceived as a way to evaluate the ability of models, most commonly ETs, to track long-range dependencies between tokens of a long sequence. The benchmark traditionally assembles six tasks:

- ListOps (1 K tokens) (Nangia and Bowman 2018)
- byte-level text classification (2 K tokens) (Maas et al. 2011)
- byte-level document retrieval (4 K tokens) (Radev et al. 2009)
- image classification on pixel-level sequences (1 K tokens) (Krizhevsky 2009)

- Pathfinder (1 K tokens) (Linsley et al. 2018) with Pathfinder-X its very long sequence variant (16 K tokens) (Tay et al. 2020)

Experimental protocol: We compare our approach to various ETs using different linearization methods from sparsification, to kernelization and low-rank approximation that are more similar to ours. We also include the results of the exponential moving average and gated Attention model Mega, and its variant Mega-chunk (Ma et al. 2022).

The aforementioned tasks propose input sequences with lengths ranging from 1 K to 16 K tokens. We align our model configuration with Nyströmformer (Xiong et al. 2021), our closest relative in the ET family of models, to be able to more accurately compare results. We consider, for the LRA benchmark, a 2 layers and 2 Attention heads configuration, with an embedding dimension of 64 for the Attention layer and an embedding dimension of 128 for the Feed Forward Network (FFN) layers. A selection of 64 indices is adopted as a baseline, again to more closely align with Nyströmformer. To obtain results in Table 1 with a Transformer model using our method (here referred to as CURformer) we run 5 experiments with random seeds and report the average accuracy.

We also note that our approach can be adapted to suit the MEGA architecture. The MEGA architecture introduces a variant of the standard Self-Attention layer using exponential moving average and gated Attention. However, the Attention matrix is present and compatible with our approach, hence we were able to create an approximated version of MEGA (here referred to as MegaCUR). Below we report results on trained versions of this modified version of MEGA and in the context of direct deployment to gauge the weight transfer compatibility, we report results on the evaluation of the model with trained weight transferred from the original MEGA model.

We test, on the LRA benchmark, selection methods based on the following per row criteria in Q and K: sum, absolute value, embedding, constant step and random. These selection methods are described in more details in Sect. 7.1.

Table 1 Comparison of results on the LRA benchmark. Higher is better. The first part of the table reports results from different ETs and a standard Transformer. The results of Mega-based architectures are in italics. Best results are in bold. Best Mega-based models' results are in italics and bold

Models	ListOps	Text	Retrieval	Image	Pathfinder	Avg
Transformer	37.11	65.21	79.14	42.94	71.83	59.24
Local Attention	36.1	60.2	76.7	40.6	66.6	56
Sparse Transformer Child et al. (2019)	17.07	63.58	72.53	44.24	71.71	53.83
Linear Attention Katharopoulos et al. (2020)	38.8	63.2	80.7	42.6	72.5	59.6
Performer Choromanski et al. (2020)	36.8	63.6	**82.2**	42.1	69.9	58.9
Linformer Wang et al. (2020)	35.70	53.94	77.83	38.56	76.34	56.47
Luna- 256 Ma et al. (2021)	37.98	65.78	79.56	**47.86**	78.55	61.95
DBA Qin et al. (2022)	38.10	66.25	80.64	46.51	**79.56**	**62.21**
Nyströmformer	37.15	65.52	79.56	41.58	70.94	58.95
FlashAttention	37.6	63.9	81.4	43.5	72.7	59.3
CURformer	**39.62**	**66.79**	78.58	45.71	73.07	59.01
Mega	***63.14***	***90.43***	***91.25***	***90.44***	***96.01***	***88.21***
Mega-chunk	*58.76*	*90.19*	*90.97*	*85.80*	*94.41*	*85.66*
MegaCUR	*61.05*	*90.19*	*90.41*	*79*	*84.41*	*75.84*

Results: We note that results in Table 1 are compiled from different sources (Qin et al. (2022), (Ma et al. 2022), Tay et al. (2020)) and different implementations. Standard Transformer and ETs in the first block are using original LRA hyperparameters (Tay et al. 2020). Nyströmformer uses a smaller architecture, described in Xiong et al. (2021) that we reproduced for our models. Mega and Mega-chunk use bigger models (in the sheer number of trainable parameters) than the standard one described in the original LRA paper to achieve the results reported in Table 1. Our results are obtained with the Abs selection method which is the best among all selection methods, as seen in Table 4.

As established in Table 1, our method compares favorably with similar methods on the LRA benchmark, achieving parity or better results. We also observe state-of-the-art results among non-Flash models, in Table 2, amounting to a speed-up of 6.65 in training time and a reduction of 91% in memory usage compared to a standard Transformer. The Flash version of our model achieves a state-of-the-art 11.3× speed-up and a 92% reduction in memory footprint. With our MegaCUR model, we find a 6.5 times speed-up and 84% reduction in memory usage while keeping most of the performance of MEGA.

Our CUR-modified MEGA model delivers results competitive with MEGA on ListOps, Text, and retrieval which all use a Softmax activation function in the MEGA gated Attention mechanism. The MEGA architecture uses the Laplace activation function for the Image and Pathfinder tasks. We applied our approach to this version as well. During training some numerical instability arises when the learning rates are too high. This instability constrains the performance of the model on the Image and Pathfinder tasks, leading to overfitting problems. We find that results cannot match the original MEGA results with 79 in accuracy in the Image task and 84.41 in Pathfinder without the optimal learning rates.

In Table 3, we report the results of direct transfer from Mega to MegaCUR on ListOps, Text, and Retrieval tasks. The accuracy results are on par with the MEGA version with its standard moving average gated Attention. Image and Pathfinder are absent because direct deployment does not work with the Laplace activation function, we discuss the potential causes of this incompatibility in Sect. 9.

Table 2 Comparison of speed and peak memory usage. For speed higher is better, and for memory lower is better. The scores of Mega-based architectures are in italics. Best scores are in bold. Best Mega-based models' scores are in italics and bold

Models	Speed ↑	Peak memory usage ↓
Transformer	1×	1×
Local Attention	5.3×	0.14×
Linear Transformer	5.6×	0.11×
Performer	5.7×	0.11×
Linformer	5.5×	0.10×
Luna- 256	4.9×	0.16×
DBA	6.1×	0.09 ×
Nyströmformer	5.6×	0.09 ×
FlashAttention	2.4×	**0.08**×
CURformer	6.65×	0.09×
FlashCURformer	**11.3**×	**0.08**×
Mega	*2.9*×	*0.31*×
Mega-chunk	*5.5*×	***0.13***×
MegaCUR	***6.5***×	*0.16*×

Table 3 Result of direct transfer from Mega to MegaCUR. Transfer is only possible for the tasks where Softmax is used in the Self-Attention of Mega. Best results are in bold

Models	ListOps	Text	Retrieval
Mega	**63.60**	90.51	**91.40**
MegaCUR Transfer	63.1	**90.53**	91.24

Table 4 Results of each method to select rows and columns in Q and K (cf. Section 7.1) on LRA, averaged over 5 runs. Best results are in bold

Selection	Sum	Abs	Embedding	Step	Random
ListOps	38.99 (± 0.63)	39.62 (± 0.62)	38.63 (± 0.49)	39.27 (± 0.59)	**39.8** (± 0.80)
Text	66.7 (± 0.06)	**66.79** (± 0.26)	66.75 (± 0.11)	66.15 (± 0.22)	66.55 (± 0.23)
Retrieval	79.12 (± 0.57)	78.85 (± 0.64)	79.20 (± 0.60)	**79.45** (± 0.83)	79.21 (± 0.62)
Image	44.68 (± 0.48)	45.71 (± 0.87)	**45.76** (± 1.49)	45.12 (± 0.50)	45.47 (± 1.37)
Pathfinder	72.61 (± 0.01)	**73.07** (± 0.008)	73.04 (± 0.006)	72.55 (± 0.009)	72.90 (± 0.00)
Avg	58.68	**59.01**	58.90	58.76	58.99

Results in Table 4 seem to indicate that attempts to target rows in Q and K to reliably increase the quality of the approximation do not bear consistently better results compared to the random baseline. This seems to indicate that all the factors accounted for in Sect. 7.1 and their impact on the approximation quality are compensated by training from scratch on the dataset.

7 Approximation analysis

To gauge the ability of our method to approximate the output of Self-Attention we compare the output of an Attention block using Self-Attention to the output of one using our method. We aim to ensure the comparison is done under realistic circumstances, to that end we use a Vision Transformer (Dosovitskiy et al. 2020) and the ImageNet- 1k dataset (Russakovsky et al. 2015).

7.1 Ablation study

We aim to perform extensive tests accounting for many factors which can impact the quality of the approximation.

The selection problem: In the absence of guarantees of approximation quality (i.e., since we do not compute A), multiple indices selection schemes were implemented to evaluate their impact on the approximation quality.

Because we aim to approximate Self-Attention through the selective computation of rows and columns of A, the targeting of rows of Q and K which would result in rows and columns of A with the highest coefficients is the most obvious approach.

The goal is then to bias selection towards salient elements of A without sacrificing inference time to compute exhaustive metrics on its rows and columns.

To do so we implemented 5 simple selection methods.

- *Sum*: Top-m selection on the row-wise sum of elements in Q and K, with m the number of indices to select.
- *Abs*: Assuming the amplitude of the values in Q and K is more important than their sign, we implement a Top-m selection on the row-wise sum of the absolute value of elements in Q and K.
- *Embed*: Top-m selection of the values found in one column of the embedding dimension of Q and K.
- *Step*: Slice of Q and K with a step defined as the result of the integer division $n \backslash m$, with n as the input sequence size and m as the number of indices to extract.
- *Random*: Selection of random indices in Q and K.

The mode of selection: The selection mode describes the way the indices are selected in Q and K. The options are to select the same indices in both or different indices in each tensor.

The linearization ratio: The number of selected indices, more precisely the ratio between this number and the size of the input sequence, will also have an impact on the quality of the approximation.

The pseudoinverse iterations: The quality of the approximation can be affected by the accuracy of the pseudoinverse, which is directly correlated with the number of iterations of the pseudoinverse algorithm.

The classification token: Finally, classification models such as ViTs use a trainable class token that aims to distill relationships between the tokens of the input sequence. In our method, constraining the selection of said token can have a great impact on the fidelity of the approximation and the resulting accuracy of the model.

Experimental protocol: The Vision Transformer base model contains 12 layers, 12 heads, with a hidden size of 768, and a total of 86 M parameters. Said model has been pre-trained on ImageNet- 21k and fine-tuned on ImageNet- 1k. We instantiate 2 versions, respectively, with standard Self-Attention and our method. The model using our method inherits the pre-trained weights of the original model. Both are fed a subset of the validation set of ImageNet- 1k, 5 images per class, accounting for a tenth of its total size, i.e. 5000 images. We cross-reference all described parameters and their influence on the accuracy of the model. To do a purely quantitative assessment of the outputs of both models we use a metric defined as

$$MEAN(|O_{attention} - O_{cur}|) \tag{4}$$

With $O_{attention}$ and O_{cur} the respective output tensors of each Self-Attention layer in both models. We compute this metric from the output of all Attention layers of the model during inference on the dataset. This metric is further contextualized by establishing its level of correlation with a known metric, the top- 1 accuracy of the model.

Results: In Fig. 5, we measure the impact of the selection method, the size of the selection and the selection of the class token on the accuracy of the model. With a baseline performance from the original Transformer of 83.9% accuracy, we can see that the best score of our approach is within 6% of the baseline performance in a direct deployment scenario. All the best scores are obtained using the class token selection. Steps 128 and 64 achieve scores of respectively 77.5% and 72.3%. Random 128 and 64 obtain the best scores of 76.8% and 66.7%. In direct deployment, the decrease in the number of indices selected has a massive impact on the accuracy of the model. The

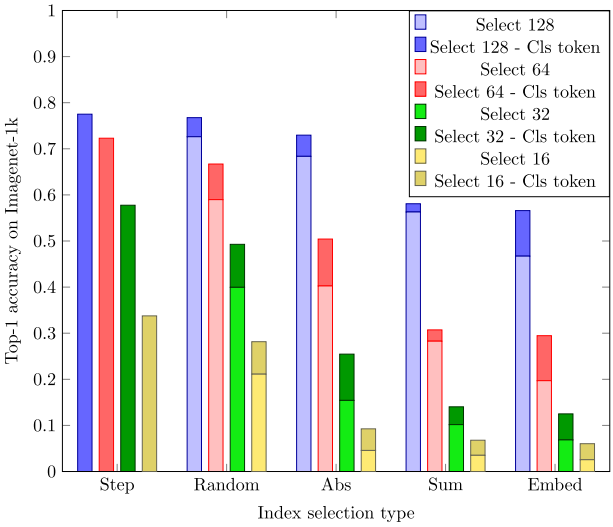

Fig. 5 Comparison of the impact of different selection parameters on the Top- 1 accuracy on Imagenet- 1k. Results are grouped by selection type, with varying number of selected tokens indicated by colors. Blue for 128, Red for 64, Green for 32 and Yellow for 16. The forced inclusion of the Cls Token is represented by the darker colors. All the selection methods indicates a similar trend, and show an increase in the score with an increase in the number of selected tokens. Forcing the inclusion of the Cls token is also always an improvement (Color figure online)

absence of the class token has an impact on performance that varies from a 5% decrease in performance for Sum 128 to a 50% decrease for Abs 16.

Figure 6 details the impact of the number of iterations of the pseudoinverse algorithm on the most performant models for all selection types. In every case, 4 iterations are

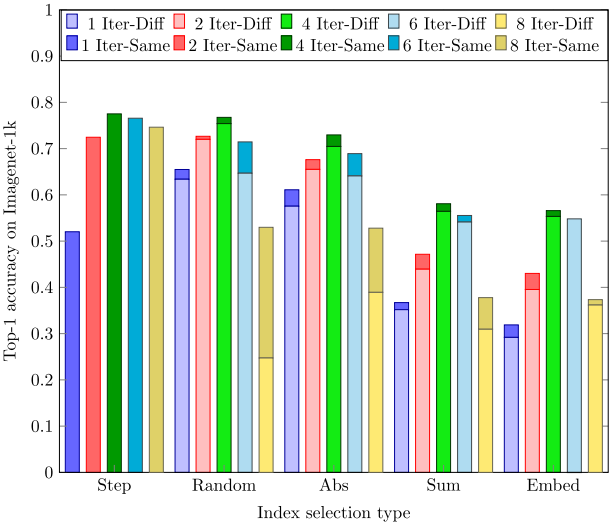

Fig. 6 Comparison of the impact of different selection parameters on the Top- 1 accuracy on Imagenet- 1k. Results are grouped by selection type, with varying number of iterations of the pseudoinverse algorithm indicated by colors. Blue for 1, Red for 2, Green for 4, Cyan for 6 and Yellow for 8. We compare two modes of selection with the Same indices selected for Q and K represented by the darker colors against lighter for different selected indices. All the selection methods take advantage of the"Same"selection mode, with improved accuracy. The number of pseudoinverse iteration appears to be a sensitive parameter, with a great decrease in score when it is too high or too low. 4 iterations giving the best results on Imagenet (Color figure online)

the optimal choice. The mode of selection also has an impact, ranging from a marginal impact to a halving of the performance of the model.

Figure 7 plots the accuracy with respect to the average difference of Self-Attention output with our method. There is a threshold beyond which the error is low enough to harness the pre-trained weights to obtain increasingly good accuracy. Before reaching that threshold, almost no increase in accuracy is recorded. Below an error of 0.4, the correlation between both metrics becomes linear until the model reaches an accuracy close to 78%.

8 Fine-tuning

To reclaim some of the lost accuracy we can fine-tune the model using our method starting from the pre-trained weights mentioned in Sect. 7.

Experimental Protocol: We keep the ViT model configuration found in Sect. 7. The model is instantiated with the standard Self-Attention model's pre-trained weights. Our Self-Attention approximation method is used in all the Transformer's layers. We further fine-tune this model on the ImageNet- 1k dataset – a one-thousand-class version of the ImageNet dataset containing 1.3 million images in its training set. The model is trained for 1532 steps using a batch of 64 images. This short training accounts for a fraction of the training set; the model is exposed to 98048 randomly selected images of the 1.3 million total images in the training set. We use this fine-tuning protocol not to obtain optimal performance, but to illustrate the low compute and time requirements to significantly increase accuracy results when transferring weights from a standard model. The evaluation score is computed on the full 50000 images of the validation set. The parameters of our Self-Attention method are the following: step selection with 4 iterations of the pseudoinverse and selection size of 16, 32, and 64.

Results: The average top- 1 accuracy, when fine-tuning on a subset of ImageNet- 1k is presented in Table 5. For each index selection amount, the model improves its accuracy. The more aggressive the linearization, the greater the accuracy gained. CUR-ViT-B16 - 384 step 16 more than doubles its accuracy in the same amount of training as CUR-ViT-B16 - 384 step 64 gains 7% in accuracy. The results from this very fast and simple fine-tuning

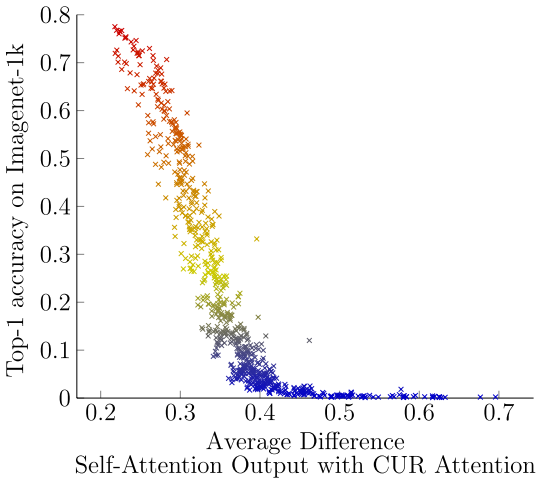

Fig. 7 Top- 1 accuracy against average difference of Self-Attention output and approximated Self-Attention output from all layers of the network

Table 5 Top- 1 accuracy on Imagenet- 1k before and after fine-tuning of ViT-B16 with approximated Self-Attention. Best results are in bold

Models	Weight Transfer	Fine-tuning
CUR-ViT-B16 - 384 step 16	33.8	70.2
CUR-ViT-B16 - 384 step 32	57.8	75.47
CUR-ViT-B16 - 384 step 64	72.3	79.35
ViT-B16 - 384	**83.9**	**83.9**

show that the loss in accuracy from the approximation after a weight transfer can be greatly decreased with low compute and time costs.

By combining the memory footprint and latency decrease with weight compatibility in the context of finetuning, all the advantages of our approach can stack up, allowing for a dramatic decrease in the time and compute necessary to perform any given task.

9 Discussion

Limitations: The performance of the approximation method is found in its ability to select relevant data from the embedded sequences in Q and K to reconstitute a good approximation of the output of standard Attention. However, the linear characteristics of the approach are derived from that approximation method and a selection of a constant number of columns and rows regardless of input sequence length. Hence, beyond a certain ratio of selected elements to the total sequence length and applied to complex tasks, requiring exhaustive context tracking, the method's performance will decrease drastically. Moreover, the results included in the main figures of the paper do not account for the best scores on all tasks. The approach is sensitive to many parameters in the column selection process; to get the best results possible for any given task, any user would need to explore those parameters to adapt the method to said task. This adds complexity to the real-world use of the method. The selection problem is still a contentious one. A less empirical and more mathematically formal approach to the problem might prove useful in finding selection methods with approximation guarantees.

The application of approximation schemes to non-Softmax Attention matrices is rarely explored in the existing literature, mostly due to the overwhelming use of Softmax in standard Transformers. We explored the capabilities of our approach in this context. Some numerical instability is present when our method is used in conjunction with the Laplace activation function. We hypothesize that this can be due to a cascading accumulation of approximation errors between the layers of the model. Softmax with its inherent normalization characteristics is better suited to endure error accumulation from layer to layer. Indeed, although bounded between 0 and 1, the coefficients obtained through the Laplace activation function do not sum to 1 in each row of the Attention matrix, unlike with Softmax. That instability can, in part, explain the ability of the Softmax version to use pre-trained weight without further fine-tuning unlike the Laplace version. However, we find that fine-tuning a model with pre-trained weight is possible with the Laplace activation function, which results in a better final accuracy (83.09) than training from scratch on the Image task in LRA.

Future work: An improved method using a variable or dynamic number of indices, selected per sequence in the batch dimension and per head in the multi-head Attention blocks, could allow for an increase in memory efficiency. Through algorithmic means

or learnable parameters, the method could identify the degree of linearization necessary to apply to any given input in both the above-mentioned dimensions. Because Attention seems to display patterns with different levels of abstraction and coherence depending on the depth of the Attention head, it is conceivable that as the pattern increases in coherence, fewer indices could be needed to obtain a good approximation of Attention. Further efforts in mitigating the instabilities found in Attention methods using non-Softmax activation functions can yield better results on a few tasks.

10 Conclusion

Transformers have established themselves as state-of-the-art, and their increasing use on longer sequences has highlighted the need to mitigate their high compute and memory costs. The emergence of efficient Transformers in the past few years has seen multiple methods aiming to mitigate the massive cost in time and memory that would ensue. In this paper, we presented such a method inspired by the CUR matrix approximation algorithm. Conceived as a drop-in replacement to standard Attention, it endeavors to efficiently approximate Attention while retaining the versatility of standard Attention. We demonstrate that it shows long-range dependency tracking ability on par or better than classical Transformers and other similar efficient approaches, can use already available pre-trained weight for fine-tuning or transfer learning, and often attains competitive results, as well as allows for direct deployment with no additional training incurring some accuracy loss depending on task complexity. When combined with the Exponential Moving Average Gated Softmax Attention of the MEGA architecture, our approach maintains similar performance and suffers no significant loss of performance in the context of weight transfer with no further training on the few tasks we tested. We harness our method's similarity with standard Self-Attention to create similar hardware-aware approaches to increase computation speed and further decrease its memory footprint. Our approximated FlashAttention implementation is faster than FlashAttention while mirroring its linear memory footprint.

Acknowledgements Author Damien François acknowledges financial support of the Institute for Advanced Studies of the University of Luxembourg through the IDAE Audacity Grant (AUDACITY- 2021)

Author contributions D.F. and M.S. wrote the first version of the manuscript.All authors participated in the redaction of every revision of the manuscript.All authors reviewed the manuscript.

Data availability >https://github.com/google-research/long-range-arena, https://www.image-net.org/download.php

Declarations

Conflict of interest The authors declare no competing interests.

References

Beltagy, I., Peters, M.E., & Cohan, A. (2020). Longformer: The long-document transformer. arXiv preprint arXiv:2004.05150

Child, R., Gray, S., Radford, A., & Sutskever, I. (2019). Generating long sequences with sparse transformers. arXiv preprint arXiv:1904.10509

Choromanski, K., Likhosherstov, V., Dohan, D., Song, X., Gane, A., Sarlos, T., Hawkins, P., Davis, J., Mohiuddin, A., & Kaiser, L., et al. (2020). Rethinking attention with performers. arXiv preprint arXiv:2009.14794

Dai, Z., Yang, Z., Yang, Y., Carbonell, J.G., Le, Q.V., & Salakhutdinov, R. (2019). Transformer-xl: Attentive language models beyond a fixed-length context. In: Annual Meeting of the Association for Computational Linguistics. https://api.semanticscholar.org/CorpusID:57759363

Dao, T. (2023). FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning. ArXiv **abs/2307.08691**

Dao, T., Fu, D.Y., Ermon, S., Rudra, A., & R'e, C. (2022). Flashattention: Fast and memory-efficient exact attention with io-awareness. ArXiv **abs/2205.14135**

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., & Houlsby, N. (2020). *An image is worth 16x16 words: Transformers for image recognition at scale.,* ArXiv **abs/2010.11929**

Drineas, P., Mahoney, M. W., & Muthukrishnan, S. (2007). Relative-error cur matrix decompositions. *SIAM Journal on Matrix Analysis and Applications, 30*, 844–881.

Ho, J., Kalchbrenner, N., Weissenborn, D., & Salimans, T. (2019). Axial attention in multidimensional transformers. arXiv preprint arXiv:1912.12180

Hu, E.J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., & Chen, W. (2021). LoRA: Low-Rank Adaptation of Large Language Models. https://arxiv.org/abs/2106.09685

Katharopoulos, A., Vyas, A., Pappas, N., & Fleuret, F. (2020). Transformers are rnns: Fast autoregressive transformers with linear attention. In: International Conference on Machine Learning, pp. 5156–5165. PMLR

Kitaev, N., Kaiser, L., & Levskaya, A. (2020). Reformer: The efficient transformer. ArXiv **abs/2001.04451**

Krizhevsky, A. (2009). Learning multiple layers of features from tiny images.

Linsley, D.A., Kim, J., Veerabadran, V., & Serre, T. (2018). Learning long-range spatial dependencies with horizontal gated-recurrent units. ArXiv **abs/1805.08315**

Ma, X., Kong, X., Wang, S., Zhou, C., May, J., Ma, H., & Zettlemoyer, L. (2021). Luna: Linear unified nested attention. ArXiv **abs/2106.01540**

Ma, X., Zhou, C., Kong, X., He, J., Gui, L., Neubig, G., May, J., & Zettlemoyer, L. (2022). Mega: Moving average equipped gated attention. ArXiv **abs/2209.10655**

Maas, A.L., Daly, R.E., Pham, P.T., Huang, D., Ng, A., & Potts, C. (2011). Learning word vectors for sentiment analysis. In: Annual Meeting of the Association for Computational Linguistics

Nangia, N., & Bowman, S.R. (2018). Listops: A diagnostic dataset for latent tree learning. ArXiv **abs/1804.06028**

Parmar, N., Vaswani, A., Uszkoreit, J., Kaiser, L., Shazeer, N., Ku, A., & Tran, D. (2018). Image transformer. In: International Conference on Machine Learning, pp. 4055–4064. PMLR

Peng, H., Pappas, N., Yogatama, D., Schwartz, R., Smith, N.A., & Kong, L. (2021). Random feature attention. arXiv preprint arXiv:2103.02143

Peng, B., Quesnelle, J., Fan, H., & Shippole, E. (2023). YaRN: Efficient Context Window Extension of Large Language Models. https://arxiv.org/abs/2309.00071

Penrose, R. (1955). A generalized inverse for matrices. *Mathematical Proceedings of the Cambridge Philosophical Society, 51*, 406–413.

Qin, B., Li, J., Tang, S., & Zhuang, Y. (2022). Dba: Efficient transformer with dynamic bilinear low-rank attention. ArXiv **abs/2211.16368**

Qiu, J., Ma, H., Levy, O., Yih, S., Wang, S., & Tang, J. (2019). Blockwise self-attention for long document understanding. ArXiv **abs/1911.02972**

Radev, D. R., Muthukrishnan, P., Qazvinian, V., & Abu-Jbara, A. (2009). The acl anthology network corpus. *Language Resources and Evaluation, 47*, 919–944.

Roy, A., Saffar, M., Vaswani, A., & Grangier, D. (2021). Efficient content-based sparse attention with routing transformers. *Transactions of the Association for Computational Linguistics, 9*, 53–68.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., & Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV), 115*(3), 211–252. https://doi.org/10.1007/s11263-015-0816-y

Tay, Y., Dehghani, M., Abnar, S., Shen, Y., Bahri, D., Pham, P., Rao, J., Yang, L., Ruder, S., & Metzler, D. (2020). Long range arena: A benchmark for efficient transformers. ArXiv **abs/2011.04006**

Tillet, P., Kung, H.-T., & Cox, D.D. (2019). Triton: an intermediate language and compiler for tiled neural network computations. Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł, & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems,30*.

Wang, S., Li, B.Z., Khabsa, M., Fang, H., & Ma, H. (2020). Linformer: Self-attention with linear complexity. ArXiv **abs/2006.04768**

Wu, Y., Kan, S., Zeng, M., & Li, M. (2023). Singularformer: Learning to decompose self-attention to linearize the complexity of transformer. In: Elkind, E. (ed.) Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23, pp. 4433–4441. International Joint Conferences on Artificial Intelligence Organization. https://doi.org/10.24963/ijcai.2023/493 . Main Track.

Xiong, Y., Zeng, Z., Chakraborty, R., Tan, M., Fung, G. M., Li, Y., & Singh, V. (2021). Nyströmformer: A nyström-based algorithm for approximating self-attention. *Proceedings of the ... AAAI Conference on Artificial Intelligence. AAAI Conference on Artificial Intelligence,35*(16), 14138–14148.

Authors and Affiliations

Damien François[1] · Mathis Saillot[3] · Jacques Klein[1] · Tegawendé F. Bissyandé[1] · Alexander Skupin[2,4,5]

✉ Damien François
damien.francois@uni.lu

✉ Mathis Saillot
mathis.saillot@univ-lorraine.fr

Jacques Klein
jacques.klein@uni.lu

Tegawendé F. Bissyandé
tegawende.bissyande@uni.lu

Alexander Skupin
alexander.skupin@uni.lu

[1] SnT, University of Luxembourg, 6 Rue Richard Coudenhove-Kalergi, 1359 Kirchberg, Luxembourg, Luxembourg

[2] LCSB, University of Luxembourg, 6 Av. du Swing, 4367 Belvaux, Luxembourg

[3] LGIPM, Université de Lorraine, 3 rue Augustin Fresnel, 57070 Metz, France

[4] Department of Physics and Materials Science, University of Luxembourg, 162A Av. de la Faiencerie, 1511 Luxembourg, Luxembourg

[5] Department of Neurosciences, University of California San Diego, 9500 Gilman Drive, 92093 La Jolla, California, USA