

This is the peer reviewed version of the following article:

DCT-Former: Efficient Self-Attention with Discrete Cosine Transform / Scribano, C.; Franchini, G.; Prato, M.; Bertogna, M. - In: JOURNAL OF SCIENTIFIC COMPUTING. - ISSN 1573-7691. - 94:(2023), pp. 1-25. [10.1007/s10915-023-02125-5]

*Terms of use:*

The terms and conditions for the reuse of this version of the manuscript are specified in the publishing policy. For all terms of use and more information see the publisher's website.

19/11/2024 06:20

(Article begins on next page)

# DCT-Former: Efficient Self-Attention with Discrete Cosine Transform

Carmelo Scribano<sup>1,2\*†</sup>, Giorgia Franchini<sup>1\*†</sup>, Marco Prato<sup>1</sup>  
and Marko Bertogna<sup>1</sup>

<sup>1</sup>Department of Physics, Informatics and Mathematics,  
University of Modena and Reggio Emilia, Modena, Italy.

<sup>2</sup>Department of Mathematical, Physical and Computer Sciences,  
University of Parma, Parma, Italy.

\*Corresponding author(s). E-mail(s):

[carmelo.scribano@unimore.it](mailto:carmelo.scribano@unimore.it); [giorgia.franchini@unimore.it](mailto:giorgia.franchini@unimore.it);

Contributing authors: [marco.prato@unimore.it](mailto:marco.prato@unimore.it);

[marko.bertogna@unimore.it](mailto:marko.bertogna@unimore.it);

†These authors contributed equally to this work.

## Abstract

Since their introduction the Trasformer architectures emerged as the dominating architectures for both natural language processing and, more recently, computer vision applications. An intrinsic limitation of this family of “fully-attentive” architectures arises from the computation of the dot-product attention, which grows both in memory consumption and number of operations as  $O(\mathbf{n}^2)$  where  $\mathbf{n}$  stands for the input sequence length, thus limiting the applications that require modeling very long sequences. Several approaches have been proposed so far in the literature to mitigate this issue, with varying degrees of success. Our idea takes inspiration from the world of *lossy* data compression (such as the JPEG algorithm) to derive an approximation of the attention module by leveraging the properties of the Discrete Cosine Transform. An extensive section of experiments shows that our method takes up less memory for the same performance, while also drastically reducing inference time. Moreover, we assume that the results of our research might serve as a starting point for a broader family of deep neural models with reduced memory footprint. The implementation will be made publicly available at <https://github.com/cscribano/DCT-Former-Public>.

**Keywords:** Transformers, Self-attention, Natural language processing, Deep learning, Discrete cosine transform, Frequencies domain

## 1 Introduction

Transformers are a family of recently introduced Deep Learning (DL) models which leverage the mechanism of dot-product attention to map a sequence of tokens of arbitrary length into a new set of tokens. Thanks to their outstanding performance in a variety of tasks, transformers are nowadays ubiquitous in state-of-the-art techniques that gain any benefit from modeling long-term interactions between elements of a sequence. Another important advantage of transformers is the ability to process sequences of arbitrary length in a single forward pass without incurring the limitations of recurrent approaches: no other standard Machine Learning (ML) or DL methods in the literature have shown this great adaptability so far. In the domain of Natural Language Processing (NLP) transformers are pervasive in any sort of task, such as Machine Translation [1–4], text classification, document retrieval, document summarization and several others more. More recently, researchers started to focus on exploiting the benefits of the self-attention mechanism for computer vision tasks [5–7], either standalone or applied downstream to a convolutional backbone and even to multimodal problems where the language and visual input needs to be correlated.

Despite the clear benefits that were widely popularized by the recent achievements, the main limitation of this class of models arises from the increase in both memory occupation and computational cost, which grows quadratically with the length of the input sequence. This problematic poses a significant limitation to the application of attention models to process long sequences. The quadratic growth in memory occupation, in particular, imposes an upper-bound on the maximum length on the sequence that can be processed. While a multitude of approaches has already been proposed in the literature to mitigate this issue, ideally aiming at making the cost of the attention grow *linearly* with the input’s length, the formulation of those solutions is often obscure and poorly interpretable.

In this work, we investigate a method to mitigate the problem of the quadratic dependence on the input’s length through the use of Discrete Cosine Transform (DCT)[8]. The DCT, widely used in signal approximation problems and especially in image compression, is well known as the most used linear transform for lossy compression. In our work, we employ DCT to compute an approximation of the real attention and to exploit such compressed representation as a replacement for the full attention. Our methodology, contrarily to other approaches, has a simple formulation and can be clearly interpreted as a mere signal filtering operation. Moreover, the proposed relaxations to the attention’s formulation can be experimentally validated against a best-case scenario formulation. We evaluate our methodology both in terms of

algorithmic complexity and in terms of performances in a common NLP benchmarking scenario. In particular, we follow the standard approach of pre-training on a large corpus of unlabeled text in an unsupervised fashion and then finetuning on downstream supervised tasks, considering the problem of sentiment classification [9] as our benchmark. It must be clear by now that the objective is not to propose a new model for language modeling tasks to compete against the state of the art. Based on a robust mathematical tool such as DCT, the mathematical treatment is also robust and the potential applications of the method are numerous.

Our contribution can be summarized as follow:

1. We propose a simple yet effective self-attention approximation by leveraging the properties of the DCT.
2. We experimentally show that our formulation allows for both reduced memory footprint and faster inference, while still being competitive on NLP tasks.
3. We compare our method against prominent competitors in the literature, showing that our method offers the best trade-off between inference time and model accuracy.

Structure of the paper: in Section 2 we introduce the main mathematical models and their formulations. In Section 3 we present an overview of the prominent methods in the literature whose purpose is to reduce the computational complexity of the mechanism of self-attention. Additionally, we present a small overview of methods that leverage Fourier-affine transforms in deep learning models. Subsequently, in Section 4 we introduce our proposed methodology to approximate the self-attention with *quasi*-linear cost. Finally, in Section 5 we detail our experimental setup and we report and discuss an experimental evaluation of the proposed methodology in multiple NLP applications.

## 2 Background

### 2.1 Neural Networks and Sequential models

Feed-Forward (FF) Artificial Neural Networks (ANN) are the simplest kind of DL model [10]. A standard FF network is a nonlinear function  $\Psi : \mathbb{R}^d \rightarrow \mathbb{R}^k$ , which maps an input  $x \in \mathbb{R}^d$  into an output  $y \in \mathbb{R}^k$ . In general, the function  $\Psi$  takes the form of a stack of Fully-Connected *layers*. Each layer is defined by a weights matrix  $W_i \in \mathbb{R}^{k_{i-1} \times k_i}$  plus a scalar bias term  $b_{i-1}$  with  $i = 1, \dots, L$ , where  $L$  is the number of layers and  $k_0 = d, k_L = k$ . Each layer is also characterized by a nonlinear activation function  $\sigma_i(\cdot)$ . The recursive formula takes the form:

$$y_i = \sigma_i(W_i^T y_{i-1} + b_i)$$

where  $y_0 = x$  and  $y_L = y$ .

On the other hand, the total function can be written as:

$$\Psi(x) = \sigma_L(W_L^T \sigma_{L-1}(W_{L-1}^T \dots \sigma_0(W_0^T x + b_0)) + b_L).$$

An important limitation of the FF model is the inability to operate with inputs of non-fixed length, which would be desirable to work with sentences in natural language and other kinds of sequential data points. In NLP, an input sentence is usually split into a set of tokens, which represents individual dictionary indices, and each token is mapped to an *embedding* of fixed dimension. For convenience here, we use the terms token and embedding interchangeably, since it persists a 1 : 1 mapping.

In contrast, Recurrent Neural Networks (RNN) [11, 12] arise expressively for the management of sequences, whether they are sequence connected by a temporal component (e.g., time series) or meaning (e.g., a sentence). An RNN processes a sequence of inputs  $X \in \mathbb{R}^{n \times d}$  by feeding sequentially each row element of  $X$   $x_i \in \mathbb{R}^d$ , referred to as a *token*, to a stack of recurrent cells. In this case, the recurrent formula is:

$$y_i(t) = \sigma_i(W_i^T y_{i-1} + \overline{W}_i^T y_i(t-1) + b_i) \quad t = (0, 1, \dots, n)$$

where  $\overline{W} \in \mathbb{R}^{k_{i-1} \times k_i}$  is a weight matrix trained during the epochs.

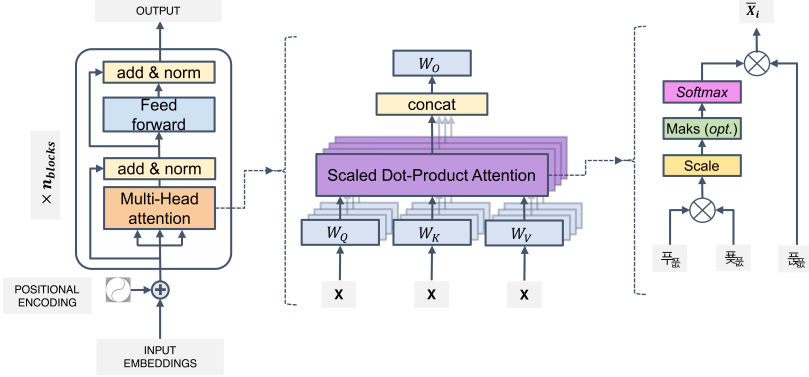
This implies that the  $i$ -th activation for the  $t$ -th token will also depend on the activation produced by the input provided at a previous timestep, effectively allowing to consider  $y_i(t)$  as a fixed size latent representation of the whole input sequence up to  $t$ . The sequential nature of the recurrent cell however poses a severe performance bottleneck by requiring the tokens to be fed to the model one after the other. Such throughput limitation, together with several other problems in terms of expressive capacity, are among the reasons that lead to the introduction of attentive and ultimately fully-attentive models.

## 2.2 Transformers

Transformers [1] represent the current state-of-the-art in DL models for sequence modeling tasks. This family of architectures replaces the recursion mechanism of RNNs with the introduction of the mechanism of *self-attention* to effectively process sequences of arbitrary length in a single forward operation. As in Figure 1, a standard transformer is made of  $\mathbf{n}_{\text{blocks}}$  identical blocks, each composed of two sub-blocks: a self-attention module and a feed-forward layer, each one followed by a layer normalization [13] operation and a residual connection.

### *Self attention*

Given a set of tokens  $X \in \mathbb{R}^{n \times d}$ , the self-attention mechanism produces a similar set of tokens  $\mathcal{X} \in \mathbb{R}^{n \times d}$ , where each new row token element of  $\mathcal{X}$  is obtained as a weighted average of the whole original set  $X$ .



**Fig. 1** Overview of the general architecture of the standard Transformer Encoder.

The resulting weights of the attention mechanism represent the affinity degree between pairs of tokens. Such affinity is computed by first projecting, where with the term projection we mean a simple matrix multiplication,  $X$  onto a set of *Queries*  $Q \in \mathbb{R}^{n \times d_q}$ , a set of *Keys*  $K \in \mathbb{R}^{n \times d_k}$  and a set of *Values*  $V \in \mathbb{R}^{n \times d_v}$ , with three distinct projection matrices  $W_Q \in \mathbb{R}^{d \times d_q}$ ,  $W_K \in \mathbb{R}^{d \times d_k}$ , and  $W_V \in \mathbb{R}^{d \times d_v}$ . The three projections are shown below:

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V \quad (1)$$

The dot-product between  $Q$  and  $K^T$  (with  $d_q = d_k$  and  $d_v = d$  in self-attention) produces an *Energy* score between pairs of tokens, which is then normalized and fed to a nonlinear *softmax* [14] operation to obtain the final weights matrix. In this case the *softmax* operation is applied row-wise.

$$E(Q, K) = \text{softmax} \left( \frac{QK^T}{\sqrt{d_q}} \right) \quad (2)$$

The energy is finally multiplied by  $V$  in order to produce the final attention output:

$$\text{Atn}(X) = E(Q, K)V = \text{softmax} \left( \frac{XW_Q(XW_K)^T}{\sqrt{d_q}} \right) XW_V \quad (3)$$

In almost any transformer implementation a number  $\mathbf{n}_{\text{heads}}$  of self-attention heads, each with its own set of projection matrices  $W_{Q,K,V}^j$ , with  $j = 1, \dots, n_{\text{heads}}$ , are applied in parallel, defining the Multi heads Self Attention (MhSA). The output of the **multi-head** attention is obtained as a concatenation of the results of the individual attention heads, usually followed by an additional projection layer  $W_O \in \mathbb{R}^{md_v \times d}$ .

$$\mathcal{X} = \text{MhSA}(X) = [\text{Atn}_1(X) \oplus \text{Atn}_2(X) \oplus \dots \oplus \text{Atn}_M(X)]W_O$$

For the sake of simplicity, from now on we can ignore the multi-head aspect of the transformers attentions, since the problem that we investigate is not dependent on the number of attention heads but is related to a single attention term.

### *Quadratic complexity of Attention*

It is clear from (2) that being  $n$  the sequence length, the complexity of calculating the attention’s weight matrix  $E \in \mathbb{R}^{n \times n}$  is  $O(n^2)$  in both memory and time, which limits significantly the applicability of the self-attention mechanism for very long input sequences. To overcome the limitations of the quadratic dependence, several options have been already proposed in the literature, some of which are discussed in section 4.2.

## 2.3 Transformer based Language modeling

Given the property of the self-attention mechanism, since their introduction, transformers have been popularized as powerful language modelers. However, transformer-based language models are known to be extraordinarily hard to train by relying only on labeled data for supervised tasks. For this reason, the scheme of adopting a pre-training strategy, already popular in previous language modeling techniques [15, 16], has become of great importance for transformers based modeling. Pre-trained transformers can be then effectively fine-tuned for downstream supervised tasks, usually with little to none architectural changes.

Among the considerable variety of pre-trained transformer models, BERT [2] and its derivatives [17–21] have become the de facto standard for deep language modeling. The strength of this model comes from the *bidirectional* pretraining strategy, which leverages a huge amount of unlabeled text in an unsupervised fashion. From an architectural standpoint, BERT simply employs the original transformer architecture adding a WordPiece tokenizer [22] to split an input sentence in a sequence of dictionary entries, which are then mapped to token embeddings. The unsupervised pre-training is carried by simultaneously optimizing for two tasks:

- Masked Language Modeling (**MLM**), where a percentage of the input tokens is masked at random, by replacing those with a placeholder [MASK] token, and then asking the model to predict back the masked tokens.
- Next Sentence Prediction (**NSP**) task, where a pair of sentences (sentence A and sentence B) are fed together to the model, divided by a separation token [SEP], and the model is tasked to classify whether the sentence B is the actual next sentence that follows A or is a random sentence from the training corpus.

The training corpus is obtained by combining BooksCorpus [23] and English Wikipedia in order to obtain over 3,5M words of document-level corpus which include long sequences of sentence-level text required for the pre-training objectives. A major downfall of the transformers pre-training is the very large

computational power required to achieve state-of-the-art performance, with a proper training easily approaching costs in the tens of thousands dollars [24] (based on the current cloud GPU prices). For our experimental validation we trained a BERT-like model following the training recipe detailed in [25], while the language modeling ability of such a model cannot be compared with a full pre-training BERT, but it is instead perfectly suited to demonstrate the advantage of our approximated attention in a fair comparison scenario. In Section 5 we detail the experimental setup and the adopted training scheme.

## 2.4 Discrete Cosine Transform

The DCT [8] is a Fourier-related transform which expresses a finite sequence of elements (a discrete signal) in terms of a sum of cosine functions at different frequencies. Most noticeably, the DCT is both discrete and, contrary to the Discrete Fourier Transform (DFT), real-valued. DCT is invertible, with the inverse function denoted as IDCT, and enjoys the properties of *energy compaction*, concentrating the energy of the signal in few coefficients, and *decorrelation*, since the coefficients are uncorrelated to each other. Thanks to those properties, DCT is heavily used as a transformation mechanism in signal processing, and especially in *lossy* data compression algorithms such as JPEG (images) [26], MPEG (video), and MPEG Layer III or MP3 (digital audio). There are several variants of DCT, the most common, also used in this work, is the type-II DCT [27], which was also the first version of DCT.

Given a finite length sequence of  $N$  real valued elements  $\hat{x} \in \mathbb{R}^{N \times 1}$ , the Type-II DCT is a sequence  $\hat{X}$  of the same length defined as:

$$\hat{X}_K = \alpha_k \sum_{n=0}^{N-1} x_n \cos\left(\frac{\pi(2n+1)k}{2N}\right) \quad \text{for } k = 0, 1, \dots, N-1$$

$$\text{where } \alpha_k = \begin{cases} \sqrt{1/N} & \text{if } k = 0 \\ \sqrt{2/N} & \text{if } k \neq 0 \end{cases} \quad (4)$$

Since the DCT is a linear transformation, (4) can be conveniently expressed in terms of a dot-product operation between the sequence  $x$  and a transformation matrix  $D \in \mathbb{R}^{N \times N}$ . Formally,  $\hat{X} = DCT(\hat{x}) = D\hat{x}$ , where:

$$D_{n,k} = \alpha_k \cos\left(\frac{\pi(2n+1)k}{2N}\right) \quad (5)$$

Due to the normalization term  $\alpha$ , the matrix  $D$  is *orthogonal*, which makes possible to express the inverse transform as  $D^{-1} = D^T$ , hence:  $IDCT(\hat{X}) = D^T \hat{X} = \hat{x}$ , thus avoiding the high computational cost of the inverse calculation.

Generally, when we speak about lossy compression algorithms, we first compute the computing of the DCT coefficients of a signal, and then keep only a handful of the most relevant values. A simple way of computing a low-frequency



approximation would be to define a matrix  $\overline{D} \in \mathbb{R}^{M \times N}$  with  $M < N$  by keeping only  $M$  rows of the transformation matrix  $D$ .  $\overline{D}$  can be used to obtain a compressed representation  $\overline{x} \in \mathbb{R}^{M \times 1}$  by computing the forward DCT, then a lossy reconstruction of the original  $\hat{x}$  is obtainable with the inverse transform. When we are dealing with a transformation that is performed by both rows and columns, we can generalize the observations made before by using 2D-DCT. In this case, given a finite length sequence of  $N \times N$  real elements  $\hat{x} \in \mathbb{R}^{N \times N}$ , the 2D-DCT can be computed with the formula  $\hat{X} = DCT(\hat{x}) = D\hat{x}D^T$ . Using the same methodology as described above, we can generalize the compression procedure to the 2-dimensional case.

## 3 Related Works

### 3.1 Efficient Attention Heads

As briefly mentioned in the introduction, the quadratic complexity of the attention is a well-studied issue in the deep learning community. A variety of solutions have so far already been proposed, which can be roughly categorized in three classes: (i) methods that try to approximate or factorize the attention as defined in the original formulation [28–34] (ii) methods that reformulate the definition of attention (e.g., by introducing locality constraints) to avoid the complexity bottleneck [35–40] (iii) contributions which entirely remove the self-attention, usually by proposing an alternative paradigm [41–43]. Our methodology clearly falls in the first category, therefore hereafter we provide a brief description of our principal competitors, a few of which will be used for comparison in the experimental Section 5.

#### *Attention Matrix Reduction*

Reformer [30] achieves a complexity of  $O(n \log(n))$  by reducing the number of operations in the computation of  $\text{softmax}(QK^T)$  (2) introducing a local-sensitivity-hashing (LSH) mechanism. Their methodology is based on the observation that large values dominate the output of the  $\text{softmax}$  operation, hence they claim to be sufficient to only compute the largest values of the  $QK^T$  product. In Linformer [28] the authors introduce a set of learnable linear projection matrices  $\overline{e}_i (i = 1, \dots, n_{heads})$  to project  $Q$  and  $V$  in a lower dimensional space, justifying this approach with the empirical observation of the attention matrix being low-rank. Performers [29] introduce a kernelizable attention mechanism (FAVOR+) to approximate the softmax attention with a complexity of  $O(n)$ . More recently, Nyströmformer exploited the usage of the Nyström approximation which is commonly used in kernel methods to approximate the Gram matrix (positive semi-definite) with a low rank matrix. To avoid computing the full attention, the authors exploit a relaxation of the Nyström method by individually computing the softmax operation of the three decomposition sub-matrices before the dot product operation.

SOFT [32] builds on top of [31] by replacing the dot-product operation with

a Gaussian kernel, thus entirely removing the softmax operation from the formulation allowing, for a proper application of the Nyström method. Moreover, they propose a Newton-Raphson based method to approximate the pseudoinverse operation, in contrast with the less efficient Moore-Penrose pseudoinverse used in [31].

### *Differences with Model Compression*

Some readers might be familiar with some popular techniques to reduce inference cost of generic deep learning models. Among those, quantization techniques [44] rely on reduced precision arithmetic (either 8-bits integers or 16-bits floating point), pruning [45] remove less important weights or nodes from the network, and knowledge distillation [46] is a technique to transfer the knowledge of a large model in a smaller one.

The formulation detailed in this manuscript, as well as the competitors previously introduced, are not related to these compression strategies. Formulations for efficient attention focus on mitigating the issue of the quadratic complexity of the dot-product attention, while compression strategies are aimed exclusively at reducing inference times and are often tailored to the particular capabilities of the hardware used for inference [47].

## 3.2 Frequencies domain

In the frequency domain, a matrix which represents a digital image is converted from spatial domain to frequency domain. The Fast Fourier Transform is an efficient method used to convert the spatial to the frequency domain. In this paper, DCT was specifically chosen to transform attention matrix information into frequencies because of some of its characteristics: DCT operates in the real field like images, its compression capability has been demonstrated and widely used in the literature, and its matrix formulation makes its computation and the computation of its inverse particularly efficient on parallel architectures. Regarding the latter point, many works have dealt with FFT parallelization, as e.g. [48] in which the authors propose a novel and hardware-efficient architecture for power-of-two FFT processors.

In the signal processing literature there is extensive use of the DCT/FFT in "learning" problems. For example in [27] the authors proposed an efficient and flexible dictionary structure for sparse and redundant signal representation and they demonstrated the advantages of the proposed structure for 3-D image denoising. On the other hand, in [49] orthogonal and nonorthogonal dictionaries are factorized as a product of a few basic transformations to balance data representation performance and computational complexity. Also in [50] the authors work with dictionary learning, with the aim of finding a frame (called dictionary) in which some training data admits a sparse representation. The approach is demonstrated experimentally both with a factorization of the Hadamard matrix and on image denoising.

### 3.3 Neural Networks in the frequency domain

To the best of our knowledge, only few works have so far exploited Fourier-related transforms in the DL domain. A remarkable contribution is the recent F-Net [42], which entirely replaces the transformer’s self-attention with a two-dimensional Discrete Fourier Transform operation. While this might sound similar to our methodology (Section 4), it is entirely different in the formulation, since their method does not represent an approximation for the dot-product operation but rather a complete substitute. Previously [51] proposed to operate a Convolutional Neural Network (CNN) on the DCT coefficients of a JPEG compressed image to avoid the need to run the full JPEG decoding algorithm. Several other contributions, such as [52–55], explored similar concepts for computer vision problems with varying degrees of success.

## 4 Proposed Methodology

### 4.1 A Naive Solution

We recall that the goal of our investigation is to exploit the DCT introduced in Section 2.4 to define an approximation method which avoids a quadratic growth of the attention matrix in (2) with the input sequence length  $n$  for an input  $X \in \mathbb{R}^{n \times d}$ .

Given the three  $(n \times d)$  matrices  $Q$ ,  $K$  and  $V$  defined in (1), all functions of input  $X$ , a straightforward solution is to individually obtain three compressed representations  $\overline{Q}$ ,  $\overline{K}$  and  $\overline{V}$  each of length  $\overline{n} \ll n$  by computing the DCT of each matrix over the dimension  $n$  and retaining only  $\overline{n}$  DCT coefficients. For ease of understanding we can express the forward DCT relying on the matrix formulation of (5), hence a transformation matrix  $\overline{D} \in \mathbb{R}^{\overline{n} \times n}$  can be easily obtained from the definition to compute the required  $\overline{n}$  DCT coefficients.

Denoting the transformation matrix as  $\overline{D}$  we formulate:

$$\overline{Q} = \overline{D}Q, \quad \overline{K} = \overline{D}K, \quad \overline{V} = \overline{D}V \quad (6)$$

by substituting in (2) we obtain:

$$\overline{E}(\overline{Q}, \overline{K}) = \text{softmax} \left( \frac{(\overline{D}Q)(K^T \overline{D}^T)}{\sqrt{d}} \right)$$

If we consider the numerator inside the softmax operator is clear to see that:

$$\overline{D}(QK^T)\overline{D}^T = DCT_{2D}(QK^T)$$

By leveraging the associative property of the dot-product,  $\overline{E} \in \mathbb{R}^{\overline{n} \times \overline{n}}$  is obtained without explicitly computing the original  $E \in \mathbb{R}^{n \times n}$ . Going forward,

the compressed attention output is computed by multiplying with  $\overline{V}$ :

$$\overline{Atn}(X) = \overline{E}(\overline{Q}, \overline{K})\overline{V} \quad (7)$$

And finally, the resulting approximated attention  $\widetilde{Atn}(X)$  is obtained with an inverse DCT:

$$\widetilde{Atn}(X) = IDCT(\overline{Atn}(X)) = (\overline{D})^T \overline{Atn}(X) \quad (8)$$

From (7), (8):

$$\widetilde{Atn}(X) = (\overline{D})^T [\overline{E}(\overline{Q}, \overline{K})] \overline{D} V = IDCT_{2D}(\overline{E}) V$$

To reiterate, our approximated attention grows in memory and complexity with  $O(\overline{n}^2)$ , by picking an  $\overline{n}$  small enough is possible to approach a linear growth with the original input length  $n$ . A clear relaxation in our method is that we are in effect leveraging:

$$\widetilde{Atn}(X) = IDCT_{2D}(softmax(DCT_{2D}(QK^T)))V \quad (9)$$

where the normalization term  $\sqrt{d_e}$  is omitted. Clearly  $softmax(DCT(x)) \neq DCT(softmax(x))$ , hence by computing the Inverse DCT we are implicitly introducing a relaxation. Similar relaxations involving the softmax function have already been proposed in [28] and [31], in Section 5.3 we discuss in detail its implications and define a strategy to experimental evaluate the performance degradation caused by its utilization.

## 4.2 A More efficient formulation

A first improvement that we can introduce to make our formulation more efficient from a computational standpoint is to avoid the calculation of three distinct forward DCT transforms as in (6). Recalling the formulation for  $Q, K$  and  $V$  in (1), we can save on computation by computing only the DCT of  $X$  ( $\overline{X} = \overline{D}X$ ) and then utilize the compressed  $\overline{X}$  in place of  $X$  in the attention formulation of (3). This can be easily proven to be equivalent to the approximated attention defined in (7). Our formulation can be then formalized as in Algorithm 1.

From the efficiency standpoint, the choice of the matrix formulation to compute the DCT is optimal: a single  $\overline{D}$  can be precomputed, memorized and shared across all the attention modules of the transformer architecture of choice. This for example in stark contrast with [28] where each attention head requires its own learnable projection matrix, resulting in a total of  $(n_{heads} * n_{blocks})$  matrices to be stored in memory. Moreover, relying on a known linear transformation matrix has its own set of advantages: (i) it reduces the total number of trainable parameters, making for a lighter and more efficient training (ii) learning a transformation matrix  $E \in \mathbb{R}^{n \times \overline{n}}$  as in [28] implies

---

**Algorithm 1** Efficient attention with DCT

---

**Input**  $X \in \mathbb{R}^{n \times d}$ **Output**  $\tilde{X} \approx \text{Atn}(X)$ **Require:**  $\overline{D} \in \mathbb{R}^{n \times n}$ 

1:  $\overline{X} = \text{DCT}(X) = \overline{D}X$

2:  $\overline{Q} = \overline{X}W_Q, \quad \overline{K} = \overline{X}W_K, \quad \overline{V} = \overline{X}W_V$

3:  $\text{Atn}(\overline{Q}, \overline{K}, \overline{V}) = E(\overline{Q}, \overline{K})\overline{V} = \text{softmax}\left(\frac{\overline{Q}\overline{K}^T}{\sqrt{d}}\right)\overline{V}$

4:  $\tilde{X} = \overline{D}^T [\text{Atn}(\overline{Q}, \overline{K}, \overline{V})]$

5: **return**  $\tilde{X}$ 

---

that the input sequence length must be exactly  $n$ , taking away the option of model input of arbitrary lengths. When using the DCT instead we can easily recompute  $\overline{D}$ , or we can exploit an algorithm for fast cosine transform without explicitly relying on  $\overline{D}$ . For the latter, in the fine-tuning experiments (Section 5.2) we employed Makhoul’s algorithm [56], which leverages the Fast Fourier Transform (FFT) to efficiently compute the DCT of a  $N$ -point real valued signal.

To conclude, among the different transforms available in the literature, DCT was chosen because it is an efficient way of compressing information, it can be expressed as a matrix product, its inverse calculation is also linear and operates in the real numbers field.

### 4.3 The curse of nonlinear softmax

As introduced in Section 5.1, a significant relaxation exploited by our formulation to compute the inverse DCT of the result of a nonlinear function  $\text{softmax}(x) : \mathbb{R}^n \rightarrow \mathbb{R}^n$  applied to the result of the forward DCT, as from (9). For the sake of completeness, we recall that the softmax function is defined as:

$$\text{softmax}(x)_i = \frac{e^{x_i}}{\sum_{j=0}^n e^{x_j}} \quad i = 0, 1, \dots, n$$

This function is commonly used in the deep learning domain to highlight larger values and hide the ones significantly smaller than the maximum; moreover, it constrains the output of a layer to sum to 1 and returns values between 0 and 1.

Ideally, to avoid our relaxation we would need a function  $\overline{\text{softmax}}$  such that:

$$\overline{\text{softmax}}(\text{DCT}(x)) = \text{DCT}(\text{softmax}(x)).$$

This function is trivially  $\overline{\text{softmax}}(\text{DCT}(x)) = \overline{\text{softmax}}(\hat{x}) = D(\text{softmax}(D^T \hat{x}))$  that is very unsuitable since it implies passing through a higher-dimensional space, which is exactly what we want to avoid with the proposed method. With our relaxation, we can instead avoid the computation of the matrix  $E \in \mathbb{R}^{n \times n}$ .

When leveraging our formulation, we introduce two potential sources of error when compared to the standard attention definition: (i) an *approximation* error induced by the lossy compression using  $\bar{n} < n$  DCT coefficients, and (ii) a *relaxation* error induced by the usage of the softmax relation above mentioned. The approximation error is intrinsically in the definition of lossy data compression, the relaxation error needs instead to be carefully evaluated in order to prove our methodology to be mathematically worthy.

---

**Algorithm 2** Evaluation of DCT-induced error
 

---

**Input**  $X \in \mathbb{R}^{n \times d}$

**Output**  $\tilde{X} \approx \text{Atn}(X)$

**Require:**  $\bar{D} \in \mathbb{R}^{\bar{n} \times n}$

1:  $Q = XW_Q, \quad K = XW_K, \quad V = XW_V$

2:  $E \leftarrow E(Q, K) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)$

3:  $\bar{E} = \bar{D}E\bar{D}^T \in \mathbb{R}^{\bar{n} \times \bar{n}}$

4:  $\tilde{E} = \bar{D}^T \bar{E} \bar{D} \in \mathbb{R}^{n \times n}$

5:  $\tilde{X} = \tilde{E}V$

6: **return**  $\tilde{X}$

---

We devise a simple yet effective strategy to experimentally evaluate the contribution of the softmax relaxation on the overall error degree: the full matrix  $E \in \mathbb{R}^{n \times n}$  is explicitly obtained as in (2), then its forward and inverse DCT are computed to obtain a lossy reconstruction  $\tilde{E} \in \mathbb{R}^{n \times n}$  which is then used in the following steps to obtain the attention output.

With this setup, the relaxed formulation of (9) is never used, hence only the approximation error is added: a simple way of quantifying the relaxation error is to compare the experimental results obtained with this formulation with those of the efficient attention formalized by Algorithm 1. It is worth clarifying that the above setup it is only intended for evaluation and comparison, since the quadratic attention is explicitly computed in Line 2 of Algorithm 2 there would not be any benefit in using this formulation in a real use scenario.

## 5 Experimental Evaluation

### 5.1 Experimental Setup

Our experimental setup follows the transfer learning scheme common in NLP: first the model is trained on a large dataset of unlabeled corpus data, then we finetune the model on a downstream supervised task. In subsection 5.3 we report the results both on the pretrain and the downstream task, while in subsection 5.2 we present the results in terms of inference speed and memory occupation.

### ***Model Architecture***

The transformer architecture adopted for our experiments is inspired by **BERT<sub>small</sub>** introduced in [57]. The model architecture follows the same structure of the original transformer [1] while only using  $n_{blocks} = 4$  instead of the 12 of *BERT<sub>base</sub>* in order to keep a reasonable memory footprint even when training with the standard attention head. Each multi-head attention uses 8 heads, the embedding dimension  $d$  is 512 and the hidden dimension of the feed-forward layer is 2048. For the input tokenization, we employed the same pretrained WordPiece tokenizer used in BERT, leveraging the implementation “*bert-base-uncased*” provided by the Transformers library [58].

### ***Pretraining***

For our evaluation, we base our workflow on the pipeline proposed [25], which combines several techniques to train a BERT-style language model with a reasonable computational budget. Following their setup, we optimize only for the masked-language model (MLM) task with a sparse token prediction head [20], not using the next sequence prediction (NSP) objective, but we used only English Wikipedia text as training corpus. To maximize the training throughput 10 masked copies of the dataset are precomputed, with a masking probability of 0.1. Moreover the maximum sequence length  $n$  is limited to 128 tokens to allow for larger batch sizes. On the optimization side, we mostly followed the same setup using the optimizer AdamW [59] with  $(\beta_1 = 0.9, \beta_2 = 0.98, \epsilon = 1e-6)$  and weight decay of 0.01. To allow for an unbiased comparison of models with vastly different training speeds, we discarded the fixed time-budget scheduler from the training recipe, instead we fixed the total number of optimization steps to  $100k$  and linearly increased the learning rate from 0 to the Peak-lr with a warm-up proportion of 0.06, then applied a linear decay for the remainder of the steps. The peak learning rate (LR) is fixed to  $1e-3$  and the minibatch size to 4096, obtained with two gradient accumulation steps.

From an implementation standpoint the optimization engine DeepSpeed [60] is used with mixed precision training provided by the APEX<sup>1</sup> backbone. To avoid potential interferences with the efficient attention formulation, we avoided using fused linear-activation-bias layers and APEX LayerNorm implementation, which are commonly used to speedup training. All our experiments are trained on two 32GB Nvidia V100 GPUs, leveraging model-level parallelism.

### ***Finetuning***

In the spirit of keeping the experimental setup simple and understandable we opted to evaluate our model on the downstream task of sentiment classification of IMDb movie reviews [9]. This dataset consists of 50.000 movie reviews in plain English text, evenly split between train and test. Each review is manually labeled for sentiment classification as positive or negative depending on the writer’s liking of the movie, positive and negative labels are distributed

---

<sup>1</sup><https://github.com/nvidia/apex>

with a 0.5 ratio both in the test and train splits, making for a perfectly balanced classification task. The sequences of the training set are in average 298 tokens long (*min.* 13, *max.* 3055), to save memory during training we cap the maximum sequence length to 1024 tokens, truncating the longer sequences.

To finetune the model, the MLM head used for pre-training is replaced by a classification head. Only the first token  $C \in \mathbb{R}^d$  is kept from the transformer’s output  $\mathcal{X} \in \mathbb{R}^{n \times d}$ .  $C$  corresponds to the special [CLS] token, which is added to the input.  $C$  is then fed to two feed-forward layers with a  $\tanh(\cdot) : \mathbb{R} \rightarrow \mathbb{R}$  activation function to produce the binary classification output. The model is optimized with a Binary CrossEntropy objective function, as for the pretraining we use the optimizer AdamW, but the learning rate is fixed to  $1e-5$ : in total we train each model for 10 epochs with a batch size of 64 with no gradient accumulation steps.

## 5.2 Evaluation

Reducing memory footprint and computational cost is the main objective of our work, therefore hereafter we provide detailed results on the requirements of our model and compare them against the main competitors in the literature. We compare our attention head against the original (*Vanilla* [1]) transformer implementation as well as Linformer, Nyströmformer and Performer.

### Model Inference results

For a fair comparison we used for all the tests our transformer model defined in Section 5.1, replacing only the attention head. We tested with randomly generated sequences of length  $n \in \{128, 512, 1024, 4096\}$  adapting the batch size accordingly to fit the model in memory: to adjust for the non fixed batch size we normalize both the inference time and the memory occupation for the current batch size. All the measurements are taken accounting only for the forward propagation.

**Table 1** Inference performance of our efficient attention compared to other attention heads

| Attention Head  | Sequence length (N) - Batch size (BS) |              |             |             |             |             |              |              |
|-----------------|---------------------------------------|--------------|-------------|-------------|-------------|-------------|--------------|--------------|
|                 | 128 - 256                             |              | 512 - 32    |             | 1024 - 16   |             | 4096 - 1     |              |
|                 | MB                                    | ms           | MB          | ms          | MB          | ms          | MB           | ms           |
| Vanilla         | 5.1                                   | 0.391        | 28.75       | 1.99        | 89.37       | 5.03        | 1250.0       | 45.6         |
| DCT-0.25        | 4.55                                  | <b>0.312</b> | 22.62       | <b>1.34</b> | <b>44.5</b> | <b>2.85</b> | <b>326.0</b> | <b>15.75</b> |
| Linformer-0.125 | <b>4.23</b>                           | 0.374        | <b>21.0</b> | 1.62        | 46.75       | 3.52        | 612.0        | 19.3         |
| Nyström-0.125   | 4.73                                  | 0.41         | 24.87       | 1.83        | 55.5        | 4.18        | 488.0        | 47.71        |
| Performer-0.125 | 4.91                                  | 0,425        | 23.87       | 1.89        | 59.5        | 4.2         | 548.0        | 28.93        |

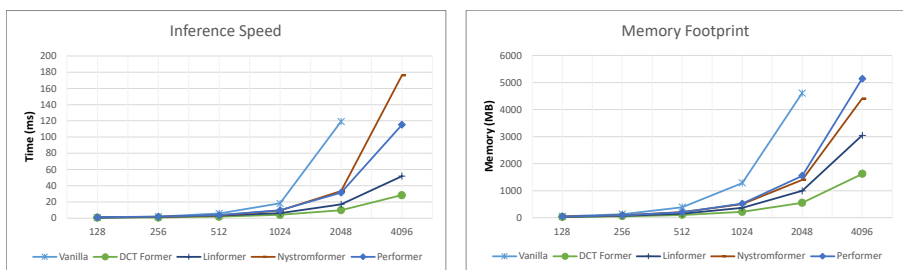
The results are expressed in terms of peak memory occupation (Megabytes, MB) and inference time (milliseconds, ms). All the models are evaluated on a single Nvidia 2080Ti



In Table 5.2 we adopt the notation  $\{\mathbf{Model}\} - \{\mathbf{scale}\}$  where **scale** indicates the (fixed) ratio of the input sequence length used to instantiate the efficient attention: for our method it defines the number of DCT coefficients, for Linformer the dimension of the learnable projection  $\bar{\epsilon}$ , for Nyströmformer the number of selected landmarks and for Performer the number of random features. While in principle scale could be defined as a constant, instead of a proportion of the input length (i.e.,  $DCT - 0.25$  for  $n = 128$  implies  $\bar{\epsilon} = 32$ ), we argue that it would be mathematically unfounded to assume that is possible to obtain a constant complexity for an arbitrary input length, whatever efficient attention head is used. From the reported results it is clear that the transformer model, equipped with our DCT based efficient attention, outperforms all the competitors. As expected and discussed in Section 2.2 the savings in memory and inference times, from the usage of our attention head, are directly proportional to the sequence length. In the next paragraph we discuss this important aspect in more details.

### Scalability with Sequence Length

To obtain the inference results presented in the last paragraph we were forced to reduce the batch-size (BS) when increasing the sequence length (N) in order to fit the model in memory. While this approach is perfectly suitable to compare different models - for a fixed sequence length - it does not allow to truly appreciate how each model scale with the sequence length. We setup a new experiment to evaluate the growth in memory occupation and inference times when we vary the sequence lengths. In fact, as reported in Figure 2 we benchmark exclusively the multi-head attention modules with a small fixed batch size. For this experiment we maintain, for each attention, the same scale factors of section 5.2.



**Fig. 2** Plot of Inference Time (ms) and Memory Footprint (MB) for input sequences of different lengths. The Batch Size is fixed to 16 to allow for comparison.

As deducible from its formulation, the vanilla attention scales proportionally with the square of the input’s length. Our efficient attention outperforms all the competitors in terms of scalability, both in inference times and memory

footprint. In particular, for the longest input sequences the benefit of the efficient attention reflects in a memory reduction of up to 80%, thus successfully enabling to work with significantly longer sequences.

In the following section we instead present the results for the trained models on both the pretraining and downstream tasks, showing that our attention can perform competitively when compared to significantly heavier formulations.

### 5.3 Results and Discussion

We evaluate multiple settings of our model following the configurations detailed in Section 5.1. For the pretraining stage, we report both the best loss (Cross Entropy) on the validation set and the Accuracy score for the MLM task. It is worth remembering that the MLM can be evaluated as a multilabel classification problem, since for each masked token of the sentence we aim at predicting the correct vocabulary entry index (which in our case is 30522 entries long). In addition, to make the comparison fair we evaluate the *normalized* accuracy score, which is obtained by dividing the accuracy score by the normalized average inference time obtained by the same model with a batch size of 256 and a sequence length of 128 (divided by  $10^2$  for readability). For the finetuning we report the averaged Precision, Recall and F1-score obtained on the test split.

All the sequences of the pretraining data are close to 128 tokens, requiring

**Table 2** Results of transformer models with different attentions

| Attention    | Pretraining       |                         |                       | Finetuning $\uparrow$ |        |          |
|--------------|-------------------|-------------------------|-----------------------|-----------------------|--------|----------|
|              | Loss $\downarrow$ | Accuracy (%) $\uparrow$ | Normalized $\uparrow$ | Precision             | Recall | F1-Score |
| Vanilla      | 2.07              | 59.7                    | 1.52                  | 0.9                   | 0.9    | 0.9      |
| DCT-16       | 2.58              | 51.6                    | 1.73                  | -                     | -      | -        |
| DCT-32       | 2.36              | 54.7                    | <b>1.74</b>           | 0.87                  | 0.87   | 0.87     |
| IDEAL-32     | 2.26              | 56.6                    | -                     | 0.88                  | 0.88   | 0.88     |
| DCT-48       | 2.28              | 56.0                    | 1.68                  | 0.86                  | 0.85   | 0.85     |
| DCT-64       | 2.24              | 56.6                    | 1.61                  | 0.85                  | 0.85   | 0.85     |
| Linformer-16 | 2.29              | 56.2                    | 1.49                  | 0.80                  | 0.80   | 0.80     |
| Linformer-32 | 2.17              | 57.9                    | 1.49                  | 0.82                  | 0.82   | 0.82     |
| Linformer-48 | 2.13              | 58.5                    | 1.49                  | 0.83                  | 0.83   | 0.83     |
| Nystrom-16   | 2.25              | 56.6                    | 1.37                  | 0.88                  | 0.87   | 0.87     |
| Nystrom-32   | 2.13              | 58.8                    | 1.26                  | 0.88                  | 0.88   | 0.88     |

For all the results the number of DCT coefficients (and respectively of Nyström landmarks and Linformer  $\bar{\tau}$  size) is fixed. We use the notation  $\{\mathbf{Model}\} - \{\bar{\mathbf{n}}\}$

only a minimal amount of padding to be fixed to exactly 128 tokens: the finetuning sequences instead, while being truncated to 1024 tokens, presents a significant length variation. For this reason we used the Makhoul’s method to compute the forward and inverse DCTs in the finetuning phase. With respect

to the usage of the  $\overline{D}$  matrix, this allows us to work with any sequence length, while otherwise we would be limited to only work with sequences of exactly  $\overline{n}$  tokens. Finetuning the Linformer it is instead far more problematic: the matrices  $\overline{\epsilon}_i$  learned during the pretraining are only suitable to work for sequences of 128, the only way to perform the finetuning is hence to reinitialize the transformation matrix to work with sequences of 1024 and zero-pad all dataset elements to the maximum length. With Nyströmformer we encountered a similar issue, since also in this case the sequence length is required to be known and be evenly divisible by the number of landmarks, hence we opted to pad the sequences in the same way as for Linformer.

For the results reported in Table 2, the models trained with the efficient attention formulation of Algorithm 1 are reported as DCT- $\{\overline{n}\}$ , while the experiment IDEAL-32 follow the formalization of Algorithm 2 to evaluate the approximation error induced by the DCT compression, without leveraging the relaxation on the softmax operation. It is fundamental to understand that, while the Ideal setup clearly outperforms the efficient setup, the Ideal setup needs to compute all the matrices onto the  $\mathbb{R}^n$  space, therefore losing all relevance to both memory and speed efficiency. Exploring alternatives to our softmax relaxation is a potentially interesting topic on its own, and can represent a future research direction.

## 6 Conclusion

In this work we analyzed the transformer architecture, in particular we focused on the attention mechanism, which grows for memory and computational time quadratically in the input length. Since in practical applications we are potentially faced with text sequences of thousands of words or videos of hundreds of frames, this growth represents the real bottleneck of these architectures. Our method, on the other hand, allows choosing the size  $\overline{n}$  of the workspace, compressing the available information through the DCT. Once we have set a compression threshold, in line with our competitors, the experiments carried out show that our method requires a memory allocation that is a quarter less than the standard attention and saves a fifth of the inference time, while still maintaining a comparable expressive capacity. Due to its great flexibility, we consider the proposed method particularly suitable for all the applications with large amounts of data. In fact, contrary to other approximations proposed in the literature, our method allows for greater adaptability and ease of applicability, by not requiring the length of the sequence to be known in advance. Then the desired memory and time usage can be chosen by defining the number of DCT coefficients to be used. As a final reminder, energy-efficiency represents a raising concern for large deep learning models: reducing inference and training cost represents one of the biggest challenges for the near future. We are confident that our work could inspire other researchers in the domain of GreenAI.

**Acknowledgments.** This work has been partially supported by the INdAM research group GNCS.

## References

- [1] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need. In: *Advances in Neural Information Processing Systems*, pp. 5998–6008 (2017)
- [2] Devlin, J., Chang, M.-W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018)
- [3] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., *et al.*: Language models are unsupervised multitask learners. *OpenAI blog* **1**(8), 9 (2019)
- [4] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J.D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., *et al.*: Language models are few-shot learners. *Advances in neural information processing systems* **33**, 1877–1901 (2020)
- [5] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., *et al.*: An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929* (2020)
- [6] Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., Zagoruyko, S.: End-to-end object detection with transformers. In: *European Conference on Computer Vision*, pp. 213–229 (2020). Springer
- [7] Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., Guo, B.: Swin transformer: Hierarchical vision transformer using shifted windows. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 10012–10022 (2021)
- [8] Ahmed, N., Natarajan, T., Rao, K.R.: Discrete cosine transform. *IEEE transactions on Computers* **100**(1), 90–93 (1974)
- [9] Maas, A.L., Daly, R.E., Pham, P.T., Huang, D., Ng, A.Y., Potts, C.: Learning word vectors for sentiment analysis. In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pp. 142–150. Association for Computational Linguistics, Portland, Oregon, USA (2011). <https://aclanthology.org/P11-1015>
- [10] Goodfellow, I.J., Bengio, Y., Courville, A.: *Deep Learning*. MIT Press, Cambridge, MA, USA (2016). <http://www.deeplearningbook.org>

- [11] Hochreiter, S., Schmidhuber, J.: Long short-term memory. *Neural computation* **9**(8), 1735–1780 (1997)
- [12] Cho, K., Van Merriënboer, B., Bahdanau, D., Bengio, Y.: On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259* (2014)
- [13] Ba, J.L., Kiros, J.R., Hinton, G.E.: Layer normalization. *arXiv preprint arXiv:1607.06450* (2016)
- [14] Bridle, J.: Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters. *Advances in neural information processing systems* **2** (1989)
- [15] Pennington, J., Socher, R., Manning, C.D.: Glove: Global vectors for word representation. In: *Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532–1543 (2014). <http://www.aclweb.org/anthology/D14-1162>
- [16] Peters, M.E., Ammar, W., Bhagavatula, C., Power, R.: Semi-supervised sequence tagging with bidirectional language models. *arXiv preprint arXiv:1705.00108* (2017)
- [17] Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., Soricut, R.: Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942* (2019)
- [18] Clark, K., Luong, M.-T., Le, Q.V., Manning, C.D.: Electra: Pre-training text encoders as discriminators rather than generators. *arXiv preprint arXiv:2003.10555* (2020)
- [19] He, P., Liu, X., Gao, J., Chen, W.: DeBERTa: Decoding-enhanced bert with disentangled attention. *arXiv preprint arXiv:2006.03654* (2020)
- [20] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., Stoyanov, V.: Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692* (2019)
- [21] Iandola, F.N., Shaw, A.E., Krishna, R., Keutzer, K.W.: SqueezeBERT: What can computer vision teach nlp about efficient neural networks? *arXiv preprint arXiv:2006.11316* (2020)
- [22] Wu, Y., Schuster, M., Chen, Z., Le, Q.V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., et al.: Google’s neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144* (2016)

- [23] Zhu, Y., Kiros, R., Zemel, R., Salakhutdinov, R., Urtasun, R., Torralba, A., Fidler, S.: Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 19–27 (2015)
- [24] Sharir, O., Peleg, B., Shoham, Y.: The cost of training nlp models: A concise overview. arXiv preprint arXiv:2004.08900 (2020)
- [25] Izsak, P., Berchansky, M., Levy, O.: How to train bert with an academic budget. arXiv preprint arXiv:2104.07705 (2021)
- [26] Raid, A.M., Khedr, W.M., El-dosuky, M.A., Ahmed, W.: Jpeg image compression using discrete cosine transform a survey. arXiv preprint arXiv:1405.6147 (2014)
- [27] Shao, X., Johnson, S.G.: Type-II/III DCT/DST algorithms with reduced number of arithmetic operations. *Signal Processing* **88**(6), 1553–1564 (2008)
- [28] Wang, S., Li, B.Z., Khabsa, M., Fang, H., Ma, H.: Linformer: Self-attention with linear complexity. arXiv preprint arXiv:2006.04768 (2020)
- [29] Choromanski, K.M., Likhoshesterov, V., Dohan, D., Song, X., Gane, A., Sarlos, T., Hawkins, P., Davis, J.Q., Mohiuddin, A., Kaiser, L., *et al.*: Rethinking attention with performers. In: International Conference on Learning Representations, p. 636 (2021)
- [30] Kitaev, N., Kaiser, L., Levskaya, A.: Reformer: The efficient transformer. In: International Conference on Learning Representations, p. 1838 (2020)
- [31] Xiong, Y., Zeng, Z., Chakraborty, R., Tan, M., Fung, G., Li, Y., Singh, V.: Nyströmformer: A Nyström-based algorithm for approximating self-attention. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 35, pp. 14138–14148 (2021)
- [32] Lu, J., Yao, J., Zhang, J., Zhu, X., Xu, H., Gao, W., Xu, C., Xiang, T., Zhang, L.: Soft: Softmax-free transformer with linear complexity. *Advances in Neural Information Processing Systems* **34**, 21297–21309 (2021)
- [33] Ren, H., Dai, H., Dai, Z., Yang, M., Leskovec, J., Schuurmans, D., Dai, B.: Combiner: Full attention transformer with sparse computation cost. *Advances in Neural Information Processing Systems* **34**, 22470–22482 (2021)
- [34] Nguyen, T., Suliafu, V., Osher, S., Chen, L., Wang, B.: Fmmformer:

- Efficient and flexible transformer via decomposed near-field and far-field attention. *Advances in Neural Information Processing Systems* **34**, 29449–29463 (2021)
- [35] Wu, C., Wu, F., Qi, T., Huang, Y., Xie, X.: Fastformer: Additive attention can be all you need. *arXiv preprint arXiv:2108.09084* (2021)
- [36] Jaszczur, S., Chowdhery, A., Mohiuddin, A., Kaiser, L., Gajewski, W., Michalewski, H., Kanerva, J.: Sparse is enough in scaling transformers. *Advances in Neural Information Processing Systems* **34**, 9895–9907 (2021)
- [37] Beltagy, I., Peters, M.E., Cohan, A.: Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150* (2020)
- [38] Tay, Y., Bahri, D., Metzler, D., Juan, D.-C., Zhao, Z., Zheng, C.: Synthesizer: Rethinking self-attention for transformer models. In: *International Conference on Machine Learning*, pp. 10183–10192 (2021)
- [39] Zhu, C., Ping, W., Xiao, C., Shoybi, M., Goldstein, T., Anandkumar, A., Catanzaro, B.: Long-short transformer: Efficient transformers for language and vision. *Advances in Neural Information Processing Systems* **34**, 17723–17736 (2021)
- [40] Chen, B., Dao, T., Winsor, E., Song, Z., Rudra, A., Ré, C.: Scatterbrain: Unifying sparse and low-rank attention. *Advances in Neural Information Processing Systems* **34**, 17413–17426 (2021)
- [41] Tolstikhin, I.O., Houlsby, N., Kolesnikov, A., Beyer, L., Zhai, X., Unterthiner, T., Yung, J., Steiner, A., Keysers, D., Uszkoreit, J., *et al.*: Mlp-mixer: An all-mlp architecture for vision. *Advances in Neural Information Processing Systems* **34**, 24261–24272 (2021)
- [42] Lee-Thorp, J., Ainslie, J., Eckstein, I., Ontanon, S.: Fnet: Mixing tokens with fourier transforms. *arXiv preprint arXiv:2105.03824* (2021)
- [43] You, W., Sun, S., Iyyer, M.: Hard-coded gaussian attention for neural machine translation. *arXiv preprint arXiv:2005.00742* (2020)
- [44] Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., Kalenichenko, D.: Quantization and training of neural networks for efficient integer-arithmetic-only inference. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2704–2713 (2018)
- [45] Han, S., Pool, J., Tran, J., Dally, W.: Learning both weights and connections for efficient neural network. In: Cortes, C., Lawrence, N., Lee,



- D., Sugiyama, M., Garnett, R. (eds.) *Advances in Neural Information Processing Systems*, vol. 28, pp. 1135–1143 (2015)
- [46] Hinton, G., Vinyals, O., Dean, J., et al.: Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* **2**(7) (2015)
- [47] Vanhoucke, V., Senior, A., Mao, M.Z.: Improving the speed of neural networks on CPUs. In: *Deep Learning and Unsupervised Feature Learning Workshop, NIPS 2011* (2011)
- [48] Zhou, Y., Noras, J.M., Shepherd, S.J.: Novel design of multiplier-less FFT processors. *Signal Processing* **87**(6), 1402–1407 (2007)
- [49] Rusu, C., Thompson, J.: Learning fast sparsifying transforms. *IEEE Transactions on Signal Processing* **65**(16), 4367–4378 (2017)
- [50] Le Magoarou, L., Gribonval, R.: Chasing butterflies: In search of efficient dictionaries. In: *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 3287–3291 (2015)
- [51] Gueguen, L., Sergeev, A., Kadlec, B., Liu, R., Yosinski, J.: Faster neural networks straight from jpeg. *Advances in Neural Information Processing Systems* **31**, 3933–3944 (2018)
- [52] Dziedzic, A., Paparrizos, J., Krishnan, S., Elmore, A., Franklin, M.: Band-limited training and inference for convolutional neural networks. In: *International Conference on Machine Learning*, pp. 1745–1754 (2019)
- [53] Rajesh, B., Javed, M., Srivastava, S., *et al.*: Dct-compcnn: A novel image classification network using jpeg compressed dct coefficients. In: *2019 IEEE Conference on Information and Communication Technology*, pp. 1–6 (2019)
- [54] Xu, K., Qin, M., Sun, F., Wang, Y., Chen, Y.-K., Ren, F.: Learning in the frequency domain. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1740–1749 (2020)
- [55] dos Santos, S.F., Sebe, N., Almeida, J.: The good, the bad, and the ugly: Neural networks straight from jpeg. In: *2020 IEEE International Conference on Image Processing (ICIP)*, pp. 1896–1900 (2020)
- [56] Makhoul, J.: A fast cosine transform in one and two dimensions. *IEEE Transactions on Acoustics, Speech, and Signal Processing* **28**(1), 27–34 (1980)
- [57] Turc, I., Chang, M.-W., Lee, K., Toutanova, K.: Well-read students learn better: On the importance of pre-training compact models. *arXiv preprint*

arXiv:1908.08962 (2019)

- [58] Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T.L., Gugger, S., Drame, M., Lhoest, Q., Rush, A.M.: Transformers: State-of-the-art natural language processing. In: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, pp. 38–45. Association for Computational Linguistics, Online (2020)
- [59] Loshchilov, I., Hutter, F.: Decoupled weight decay regularization. arXiv preprint arXiv:1711.05101 (2017)
- [60] Rasley, J., Rajbhandari, S., Ruwase, O., He, Y.: Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In: Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 3505–3506 (2020)