Efficient Transformers: A Survey

Yi Tay yitay@google.com
Google Research

Mostafa Dehghani dehghani@google.com
Google Research, Brain team

Dara Bahri dbahri@google.com
Google Research

Donald Metzler metzler@google.com
Google Research

Editor: Preprint

Abstract

Transformer model architectures have garnered immense interest lately due to their effectiveness across a range of domains like language, vision and reinforcement learning. In the field of natural language processing for example, Transformers have become an indispensable staple in the modern deep learning stack. Recently, a dizzying number of *"X-former"* models have been proposed - Reformer, Linformer, Performer, Longformer, to name a few - which improve upon the original Transformer architecture, many of which make improvements around computational and memory *efficiency*. With the aim of helping the avid researcher navigate this flurry, this paper characterizes a large and thoughtful selection of recent efficiency-flavored "X-former" models, providing an organized and comprehensive overview of existing work and models across multiple domains.

Keywords: Deep Learning, Natural Language Processing, Transformer Models, Attention Models

1. Introduction

Transformers (Vaswani et al., 2017) are a formidable force in the modern deep learning stack. Transformers are pervasive and have made tremendous impact in many fields such as language understanding (Devlin et al., 2018; Brown et al., 2020; Raffel et al., 2019) and image processing (Parmar et al., 2018; Carion et al., 2020). As such, it is only natural that a wealth of research has been dedicated to making fundamental improvements to the model over the past few years (Dehghani et al., 2018; So et al., 2019; Ahmed et al., 2017). This immense interest has also spurred research into more efficient variants of the model (Kitaev et al., 2020; Roy et al., 2020; Beltagy et al., 2020; Katharopoulos et al., 2020; Tay et al., 2020b; Wang et al., 2020b; Rae et al., 2020).

There has been such a surge of Transformer model variants proposed recently, that researchers and practitioners alike may find it challenging to keep pace with the rate of innovation. As of this writing (circa August 2020), there have been nearly a dozen new efficiency-focused models proposed in just the past 6 months. Thus, a survey of the existing literature is both beneficial for the community and quite timely.

The self-attention mechanism is a key defining characteristic of Transformer models. The mechanism can be viewed as a graph-like inductive bias that connects all tokens in a sequence with a relevance-based pooling operation. A well-known concern with self-attention is the quadratic time and memory complexity, which can hinder model scalability in many settings. There has been an overwhelming influx of model variants proposed recently that address this problem. We hereinafter name this class of models *"efficient Transformers"*.

Based on the context, *efficiency* of a model can be interpreted differently. It might refer to the memory footprint of the model, which is of importance when the memory of accelerators on which the model is running is limited. The efficiency might also refer to computational costs, e.g. number of FLOPs, both during training and inference. In particular, for on-device applications, models are supposed to be able to operate with a limited computational budget. Here in this survey, we refer to the efficiency of Transformers, both in terms of memory and computation, when they are used for modeling large inputs.

Efficient self-attention models are crucial in applications that model long sequences. For example, documents, images, and videos are all often composed of a relatively large number of pixels or tokens. Efficiency in processing long sequences is therefore paramount for widespread adoption of Transformers.

This survey sets out to provide a comprehensive overview of the recent advances made in this class of models. We are primarily interested in modeling advances and architectural innovations that improve the efficiency of Transformers by tackling the quadratic complexity issue of the self-attention mechanism, we also briefly discuss general improvements and other efficiency improvements in subsequent sections.

In this paper, we propose a taxonomy of efficient Transformer models, characterizing them by the technical innovation and primary use case. Specifically, we review Transformer models that have applications in both language and vision domains, attempting to consolidate the literature across the spectrum. We also provide a detailed walk-through of many of these models and draw connections between them.

2. Background on Transformers

This section provides an overview of the well-established Transformer architecture (Vaswani et al., 2017). Transformers are multi-layered architectures formed by stacking Transformer blocks on top of one another.

Transformer blocks are characterized by a multi-head self-attention mechanism, a position-wise feed-forward network, layer normalization (Ba et al., 2016) modules and residual connectors. The input to the Transformer model is often a tensor of shape $\mathbb{R}^B \times \mathbb{R}^N$, where B is the batch size, N the sequence length.

The input first passes through an embedding layer that converts each one-hot token representation into a d dimensional embedding, i.e., $\mathbb{R}^B \times \mathbb{R}^N \times \mathbb{R}^D$ The new tensor is then additively composed with positional encodings, and passed through a multi-headed self-attention module. Positional encodings can take the form of a sinusoidal input (as per (Vaswani et al., 2017)) or be trainable embeddings.

The inputs and output of the multi-headed self-attention module are connected by residual connectors and a layer normalization layer. The output of the multi-headed self-

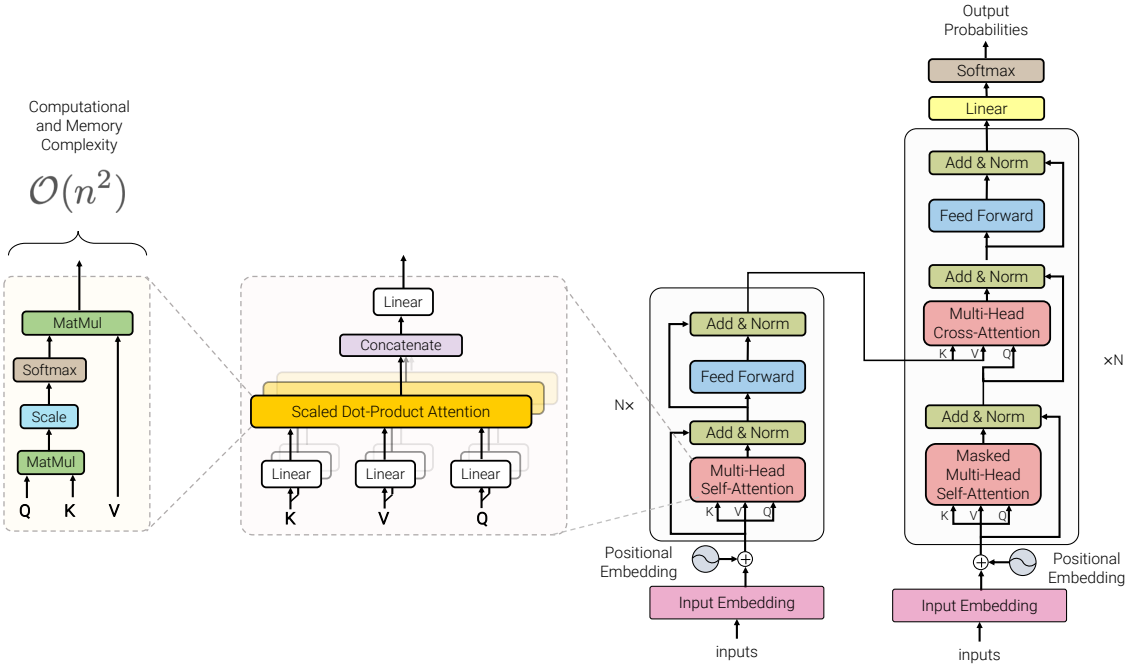


Figure 1: Architecture of the standard Transformer (Vaswani et al., 2017)

attention module is then passed to a two-layered feed-forward network which has its inputs/outputs similarly connected in a residual fashion with layer normalization. The sub-layer residual connectors with layer norm is expressed as:

$$X = \text{LayerNorm}(F_S(X)) + X$$

where F_S is the sub-layer module which is either the multi-headed self-attention or the position-wise feed-forward layers.

2.1 Multi-Head Self-Attention

The Transformer model leverages a multi-headed self-attention mechanism. The key idea behind the mechanism is to learn an alignment (Bahdanau et al., 2014) in which each element in the sequence learns to gather from other tokens in the sequence. The operation for a single head is defined as:

$$A_h = \text{Softmax}(\alpha Q_h K_h^\top) V_h,$$

where $Q_h = \boldsymbol{W}_q X, K_h = \boldsymbol{W}_k X$ and $V_h = \boldsymbol{W}_v X$ are linear transformations applied on the temporal dimension of the input sequence. $\boldsymbol{W}_q, \boldsymbol{W}_k, \boldsymbol{W}_v \in \mathbb{R}^{d \times \frac{d}{N_h}}$ are the weight matrices (parameters) for the query, key and value projections and project the input X to an output tensor of d dimensions. N_h is the number of heads

X is a matrix of size $\mathbb{R}^N \times \mathbb{R}^d$, and α is a scaling factor that is typically set to $\frac{1}{\sqrt{d}}$. Let us denote the number of heads by N_H. The outputs of heads $A_1 \cdots A_{N_H}$ are concatenated together and passed into a dense layer. The output Y can thus be expressed as $Y =$

$\boldsymbol{W}_o[A_1 \cdots A_{N_H}]$, where $\boldsymbol{W}_o$ is an output linear projection. Note that the computation of A is typically done in a parallel fashion by considering tensors of $\mathbb{R}^B \times \mathbb{R}^N \times \mathbb{R}^{N_h} \times \mathbb{R}^{\frac{d}{N_h}}$ and computing the linear transforms for all heads in parallel.

The attention matrix $A = QK^\top$ is chiefly responsible for learning alignment scores between tokens in the sequence. In this formulation, the dot product between each element/token in the query (Q) and key (K) is taken. This drives the self-alignment process in self-attention whereby tokens learn to *gather* from each other.

On the scalability of Self-Attention At this point, it is apparent that the memory and computational complexity required to compute the attention matrix is quadratic in the input sequence length, i.e., $N \times N$. In particular, the $QK^\top$ matrix multiplication operation alone consumes N^2 time and memory. This restricts the overall utility of self-attentive models in applications which demand the processing of long sequences. In subsequent sections, we discuss methods that reduce the cost of self-attention.

2.2 Position-wise Feed-forward Layers

The outputs of the self-attention module is then passed into a two-layered feed-forward network with ReLU activations. This feed-forward layer operates on each position independently hence the term *position-wise*. This is expressed as follows:

$$F_2(ReLU(F_1(X_A)))$$

where F_1, F_2 are feed-forward functions of the form $Wx + b$.

2.3 Putting it all together

Each Transformer block can be expressed as:

$$X_A = \text{LayerNorm}(\text{MultiheadSelfAttention}(X)) + X$$
$$X_B = \text{LayerNorm}(\text{PositionFFN}(X_A)) + X_A$$

where X is the input of the Transformer block and X_B is the output of the Transformer block.

2.4 Transformer Mode

It is important to note the differences in the mode of usage of the Transformer block. Transformers can primarily be used in three ways, namely: (1) *encoder-only* (e.g., for classification), (2) *decoder-only* (e.g., for language modeling), and (3) *encoder-decoder* (e.g., for machine translation). In encoder-decoder mode, there are usually multiple multi-headed self-attention modules, including a standard self-attention in both the encoder and the decoder, along with an encoder-decoder cross-attention that allows the decoder to utilize information from the encoder. This influences the design of the self-attention mechanism. In the encoder mode, there is no restriction or constraint that the self-attention mechanism has to be causal, i.e., dependent solely on the present and past tokens. In the encoder-decoder setting, the encoder and encoder-decoder cross attention can afford to be non-causal but the decoder self-attention *must* be causal. The ability to support causal auto-regressive

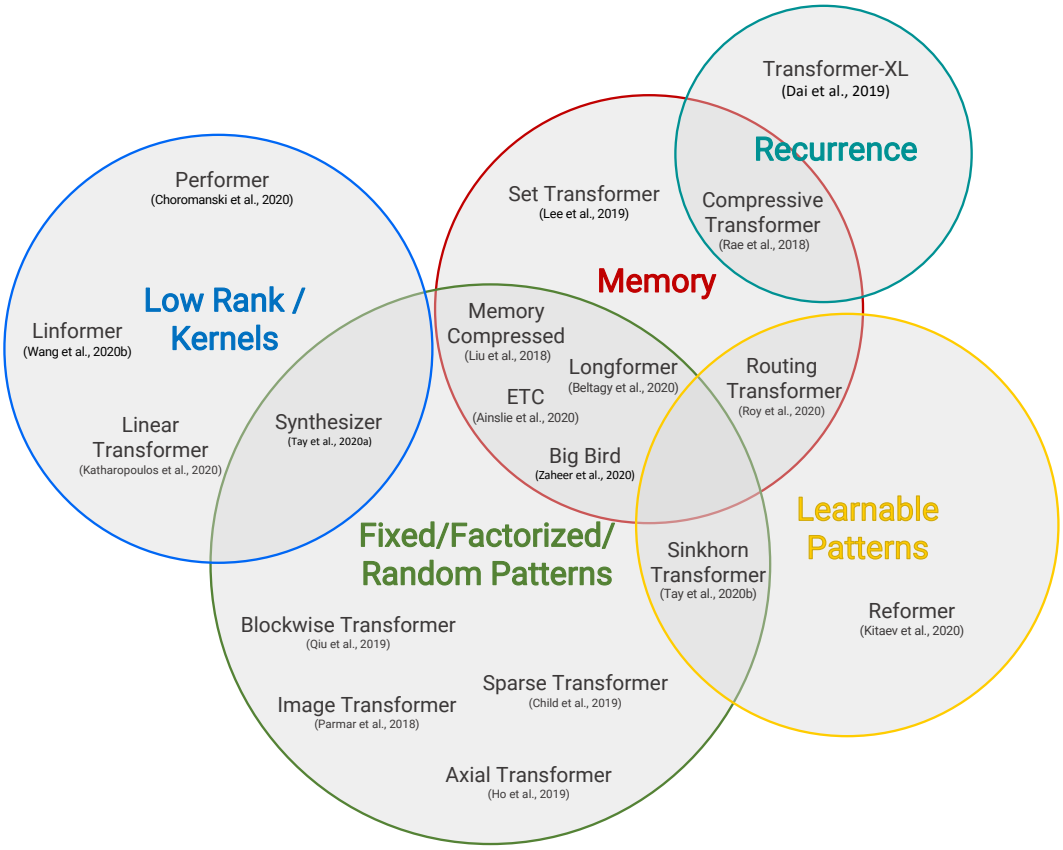


Figure 2: Taxonomy of Efficient Transformer Architectures.

decoding is required when designing efficient self-attention mechanisms since it can be a limiting factor in many applications.

3. A Survey of Efficient Transformer Models

In this section, we provide a high-level overview of efficient Transformer models. We begin by presenting a characterization of the different models. Table 1 lists the efficient Transformers released to date while Figure 2 presents a graphical overview of several key efficient Transformer models.

3.1 A Taxonomy of Efficient Transformers

This section outlines a general taxonomy of efficient Transformer models, characterized by their core techniques and primary use case. The primary goal of most of these models, with the exception of those based on segment-based recurrence, is to *approximate* the quadratic-cost attention matrix. Each method applies some notion of **sparsity** to the otherwise dense attention mechanism.

Model / Paper	Complexity	Decode	Class
Memory Compressed† (Liu et al., 2018)	$\mathcal{O}(n_c^2)$	✓	FP+M
Image Transformer† (Parmar et al., 2018)	$\mathcal{O}(n.m)$	✓	FP
Set Transformer† (Lee et al., 2019)	$\mathcal{O}(nk)$	✗	M
Transformer-XL† (Dai et al., 2019)	$\mathcal{O}(n^2)$	✓	RC
Sparse Transformer (Child et al., 2019)	$\mathcal{O}(n\sqrt{n})$	✓	FP
Reformer† (Kitaev et al., 2020)	$\mathcal{O}(n \log n)$	✓	LP
Routing Transformer (Roy et al., 2020)	$\mathcal{O}(n \log n)$	✓	LP
Axial Transformer (Ho et al., 2019)	$\mathcal{O}(n\sqrt{n})$	✓	FP
Compressive Transformer† (Rae et al., 2020)	$\mathcal{O}(n^2)$	✓	RC
Sinkhorn Transformer† (Tay et al., 2020b)	$\mathcal{O}(b^2)$	✓	LP
Longformer (Beltagy et al., 2020)	$\mathcal{O}(n(k+m))$	✓	FP+M
ETC (Ainslie et al., 2020)	$\mathcal{O}(n_g^2 + nn_g)$	✗	FP+M
Synthesizer (Tay et al., 2020a)	$\mathcal{O}(n^2)$	✓	LR+LP
Performer (Choromanski et al., 2020)	$\mathcal{O}(n)$	✓	KR
Linformer (Wang et al., 2020b)	$\mathcal{O}(n)$	✗	LR
Linear Transformers† (Katharopoulos et al., 2020)	$\mathcal{O}(n)$	✓	KR
Big Bird (Zaheer et al., 2020)	$\mathcal{O}(n)$	✗	FP+M

Table 1: Summary of Efficient Transformer Models presented in chronological order of their first public disclosure. Some papers presented sequentially may first appear at the same time, e.g., as an ICLR submission. Papers annotated with a superscript † are peer-reviewed papers. Class abbreviations include: FP = Fixed Patterns or Combinations of Fixed Patterns, M = Memory, LP = Learnable Pattern, LR = Low Rank, KR = Kernel and RC = Recurrence. Furthermore, n generally refers to the sequence length and b is the local window (or block) size. We use subscript g on n to denote global memory length and n_c to denote convolutionally compressed sequence lengths.

- **Fixed Patterns (FP)** - The earliest modifications to self-attention simply sparsifies the attention matrix by limiting the field of view to fixed, predefined patterns such as local windows and block patterns of fixed strides.
  - **Blockwise Patterns** The simplest example of this technique in practice is the blockwise (or chunking) paradigm which considers blocks of local receptive fields by chunking input sequences into fixed blocks. Examples of models that do this include Blockwise (Qiu et al., 2019) and/or Local Attention (Parmar et al., 2018). Chunking input sequences into blocks reduces the complexity from N^2 to B^2 (block size) with $B << N$, significantly reducing the cost. These blockwise or chunking methods serve as a basis for many more complex models.
  - **Strided Patterns** Another approach is to consider strided attention patterns, i.e., only attending at fixed intervals. Models such as Sparse Transformer (Child et al., 2019) and/or Longformer (Beltagy et al., 2020) employ strided or "dilated" windows.
  - **Compressed Patterns** - Another line of attack here is to use some pooling operator to down-sample the sequence length to be a form of fixed pattern. For

instance, Compressed Attention (Liu et al., 2018) uses strided convolution to effectively reduce the sequence length.

- **Combination of Patterns (CP)** - The key idea of combined[1] approaches is to improve coverage by combining two or more distinct access patterns. For example, the Sparse Transformer (Child et al., 2019) combines strided and local attention by assigning half of its heads to pattern. Similarly, Axial Transformer (Ho et al., 2019) applies a sequence of self-attention computations given a high dimensional tensor as input, each along a single axis of the input tensor. In essence, the combination of patterns reduces memory complexity in the same way that fixed patterns does. The difference, however, is that the aggregation and combinaton of multiple patterns improves the overall coverage of the self-attention mechanism.

- **Learnable Patterns (LP)** - An extension to fixed, pre-determined pattern are *learnable* ones. Unsurprisingly, models using learnable patterns aim to learn the access pattern in a data-driven fashion. A key characteristic of learning patterns is to determine a notion of token relevance and then assign tokens to buckets or clusters. Notably, Reformer (Kitaev et al., 2020) introduces a hash-based similarity measure to efficiently cluster tokens into chunks. In a simlar vein, the Routing Transformer (Roy et al., 2020) employs online k-means clustering on the tokens. Meanwhile, the Sinkhorn Sorting Network (Tay et al., 2020b) exposes the sparsity in attention weights by learning to to sort blocks of the input sequence. In all these models, the similarity function is trained end-to-end jointly with the rest of the network. The key idea of learnable patterns is still to exploit fixed patterns (chunked patterns). However, this class of methods learn to sort/cluster the input tokens - enabling a more optimal global view of the sequence while maintaining the efficiency benefits of fixed patterns approaches.

- **Memory** - Another prominent method is to leverage a side memory module that can access multiple tokens at once. A common form is *global* memory which is able to access the entire sequence. The global tokens act as a form of memory that learns to gather from input sequence tokens. This was first introduced in Set Transformers (Lee et al., 2019) as the *inducing points* method. These parameters are often interpreted as "memory" and are used as a form of *temporary* context for future processing. This can be thought of as a form of parameter attention (Sukhbaatar et al., 2019). Global memory is also used in ETC (Ainslie et al., 2020) and Longformer (Beltagy et al., 2020). With a limited number of memory (or inducing points), we are able to perform a preliminary *pooling* like operation of the input sequence to compress the input sequence - a neat trick to have at one's disposal when designing efficient self-attention modules.

- **Low-Rank Methods** - Another emerging technique is to improve efficiency by leveraging low-rank approximations of the self-attention matrix. The key idea is to assume low-rank structure in the $N \times N$ matrix. The Linformer (Wang et al., 2020b) is a

1. We note that this is also often referred to as factorization approaches, e.g., in (Child et al., 2019). We decide to refer to this class of models as combination approaches because (1) it is a better fit to what these models are actually doing and (2) to avoid confusion with matrix factorization or low-rank approaches.

classic example of this technique, as it projects the length dimension of keys and values to a lower-dimensional representation ($N \rightarrow k$). It is easy to see that the low-rank method ameliorates the memory complexity problem of the self-attention because the $N \times N$ matrix is now decomposed to $N \times k$.

- **Kernels** - Another recently popular method to improve the efficiency of Transformers is to view the attention mechanism through kernelization. The usage of kernels (Katharopoulos et al., 2020; Choromanski et al., 2020) enable clever mathematical re-writing of the self-attention mechanism to avoid explicitly computing the $N \times N$ matrix. Since kernels are a form of approximation of the attention matrix, they can be also viewed as a form of low-rank method (Choromanski et al., 2020).

- **Recurrence** - A natural extension to the blockwise method is to connect these blocks via recurrence. Transformer-XL (Dai et al., 2019) proposed a segment-level recurrence mechanism that connects multiple segments and blocks. These models can, in some sense, be viewed as *fixed pattern* models. However, we decided to create its own category due to its deviation from other block / local approaches.

We note that these buckets are broad characterization of the different efficient Transformer models. In reality, there is no sharp boundary between these groups and models may be composed of multiple different technical innovations. For example, the k-means clustering in Routing Transformer (Roy et al., 2020) can be also interpreted as a form of global memory approach, since one can view the centroids as parameterized memory. In Reformer, however, this is used to learn the sparsity pattern of the attention weights. Additionally, pooling (Liu et al., 2018) can be also interpreted as a form of memory model.

3.2 Detailed Walk-through of Efficient Transformer Models

This section delves into the details of several key efficient Transformer models, discussing their pros, cons, and unique talking points. Due to the large number of Transformer models, we sample several models to elaborate on.

Structure of this section We begin by discussing local and fixed patterns models such as the Memory Compressed Transformer (Liu et al., 2018) and Image Transformer (Parmar et al., 2018). We then discuss the Set Transformers (Lee et al., 2019), an early approach for utilizing global memory. Following which, we move on to models that utilize combinations of patterns such as Sparse Transformers (Child et al., 2019) and Axial Transformers (Ho et al., 2019). Next, we discuss Longformer (Beltagy et al., 2020) and ETC (Ainslie et al., 2020), an introduction of the memory-based approaches to the Sparse Transformer family. Our detailed walkthrough moves on to models that incorporate learnable patterns (LP) such as Routing Transformers (Roy et al., 2020), Reformer (Kitaev et al., 2020) and Sinkhorn Transformers (Tay et al., 2020b). After which, we introduce Linformer (Wang et al., 2020b) and Synthesizers (Tay et al., 2020a), models that can be considered low-rank factorization approaches. We then discuss models based on kernel approaches such as Performer (Choromanski et al., 2020) and Linear Transformers (Katharopoulos et al., 2020). Finally, we discuss the models that are based on segment-based recurrence such as Transformer-XL (Dai et al., 2019) and Compressive Transformers (Rae et al., 2020).

3.2.1 Memory Compressed Transformer

Memory Compressed Transformer (Liu et al., 2018) is one of the early attempts for modifying Transformers in order to handle longer sequences. The modification introduced in Memory Compressed Transformer is in two folds: localizing the attention span and using memory compressed attention.

Local Attention Span An straightforward solution for dealing with long sequences in Transformers is to limit the attention span to a local neighborhood. Liu et al. (2018) proposed dividing the input sequence into blocks of similar length and run self-attention within each block independently. This keeps the cost of attention per block constant, thus the number of activations scales linearly by the input length.

Memory-compressed Attention The idea of memory compressed attention is to reduce the number of keys and queries using a strided convolution, while the queries remain unchanged. This leads to reduction of the size of the attention matrix as well as the attention computations based on a compression factor that depends on the kernel size and the strides of the convolution. Memory compressed attention lets the model exchange the information globally across the input sequence as opposed to the local attention.

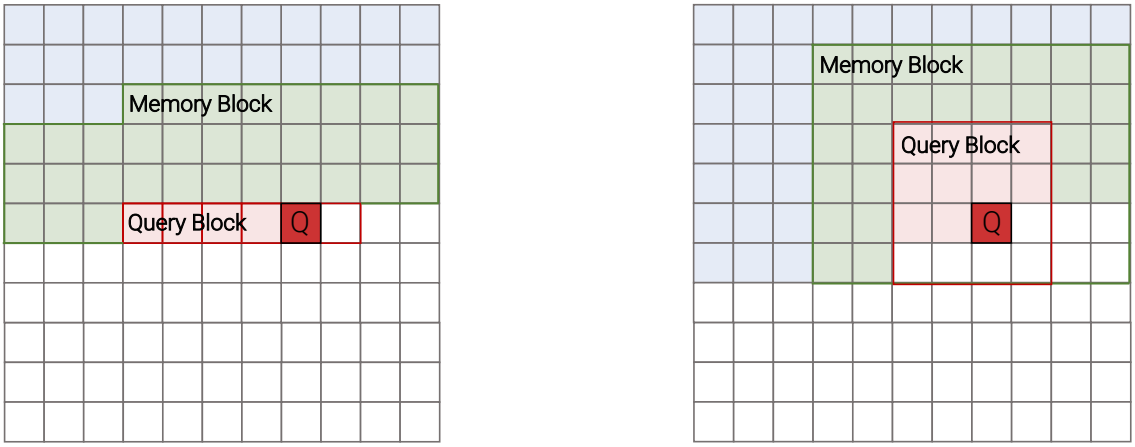
Computation and Memory Complexity For the block size of b, the computational and memory cost of self-attention is each block $\mathcal{O}(b^2)$ and given there are n/b blocks, the computational and memory cost of local attention is $\mathcal{O}(b.n)$. For the memory-compressed attention, applying a convolution with kernel size and strides of k, the computational and memory cost of the attention mechanism reduces to $\mathcal{O}(n.n/k)$.

3.2.2 Image Transformer

Image Transformer (Parmar et al., 2018), inspired by convolutional neural networks, restricts the receptive field of self-attention to only local neighborhoods. This helps the model scale up to process larger batch sizes while keeping the likelihood loss tractable. Besides the efficiency, adapting the notion of loctality can be a desirable inductive bias for processing images. Image Transformer offers the encoder-decoder architecture, where the encoder generates a contextualized representation for every pixel-channel in the inputs and the decoder autoregressively generates one channel per pixel at each time step.

Localized Attention Span Limiting the receptive field to a local neighborhood (Parmar et al., 2018, 2019) addresses the issue with the computation and memory cost of running global self-attention on large inputs, but changing the neighborhood per query position would prohibit packing the computations of the self-attention into two matrix multiplications. To avoid that, Image Transformer proposes partitioning the inputs into "query blocks and their associated "memory blocks", where for all queries from a single query block, the model attends to the same memory block. There are two different schemes for choosing query blocks and their associated memory block neighborhoods: *1-dimensional local attention* and *2-dimensional local attention*. Here we briefly explain these schemes in the decoder case.

For the 1-dimensional local attention, the image is flattened in the raster order and partitioned into non-overlapping query blocks Q of length l_q, and for each query block,



(a) 1-dimensional local attention

(b) 2-dimensional local attention

Figure 3: Attention span in Image Transformer on a two-dimensional input.

a memory block M is build from the same pixels in the Q as well as a fixed number of pixels, l_m, generated before the query pixel. In the 2-dimensional local attention, pixels are generated in raster order. For the 2-dimensional local attention, the image is partitioned into multiple non-overlapping rectangular query blocks of length $l_q = w_q \times h_q$. The memory block extends the query block to the top, left h_m and w_m pixels and to the right w_m pixels, so $l_m = (w_q \times q_h) + 2 \times (h_m + w_m)$. The query pixel can attend to all other pixels. In the 2-dimensional local attention, pixels in the image are generated one query block after another. Generated blocks are in raster order, as well as generated pixels inside every block.

Computational and Memory Complexity In Image Transformer, the attention matrix has the shape of $l_q \times m$, where l_q is the chosen length for the query blocks and M is the length of the memory block (which is in fact $l_q + l_m$). Given that memory blocks do not overlap, we have to compute $n \times l_q$ attention matrices. Thus the memory and computational complexity of Image Transformer is $\mathcal{O}(n.m)$.

Restrictions Image Transformer, and in general restricting the context in the attention mechanism to a local neighborhood, can decrease the cost of memory and computation at the price of losing the global receptive field. This can be an issue where global information is required to solve the task. Also, local-attention still has quadratic complexity to the region length, introducing an extra hyper-parameter to the trade-off between performance and computational complexity.

3.2.3 Set Transformer

The Set Transformer (Lee et al., 2019) adapts the Transformer model for *set-input* problems - that is, problems wherein the input is a set of features and the output is some function of this set (and is thereby invariant to the permutation, or ordering, of the input features). The Set Transformer leverages attention to capture interactions between elements of the input set. Furthermore, it applies the idea of *inducing points* from the sparse Gaussian

process literature to reduce the complexity of attention from quadratic to linear in the size of the input set.

Problems involving sets of objects often have a *permutation invariance* property: the target value for the set is the same regardless of the order of the objects in the set. Zaheer et al. (2017) proved that all permutation-invariant functions can be represented by the following functional form:

$$\text{network}\left(\{x_1, \ldots, x_N\}\right) = \rho\left(\text{pool}\left(\{\phi(x_1), \ldots, \phi(x_N)\}\right)\right),$$

where the pooling function pool is a simple summation and ϕ and ρ are continuous functions. This form can be interpreted as the composition of an *encoder* ϕ and *decoder* $\rho\left(\text{pool}(\cdot)\right)$. While this form is a universal approximator in the space of permutation-invariant functions, it is unclear how well such models will fit tasks in practice, given a limited capacity. The Set Transformer proposes a solution that can be viewed as a encoder + pooled decoder, but where, unlike the form given above, the encoder and decoder can attend to input elements individually and the pooling function is parameterized.

Attention Blocks The model introduces the following constructs: "Multihead Attention Block" (MAB), "Set Attention Block" (SAB), "Induced Set Attention Block" (ISAB), and "Pooling by Multihead Attention" (PMA). They are defined as follows.

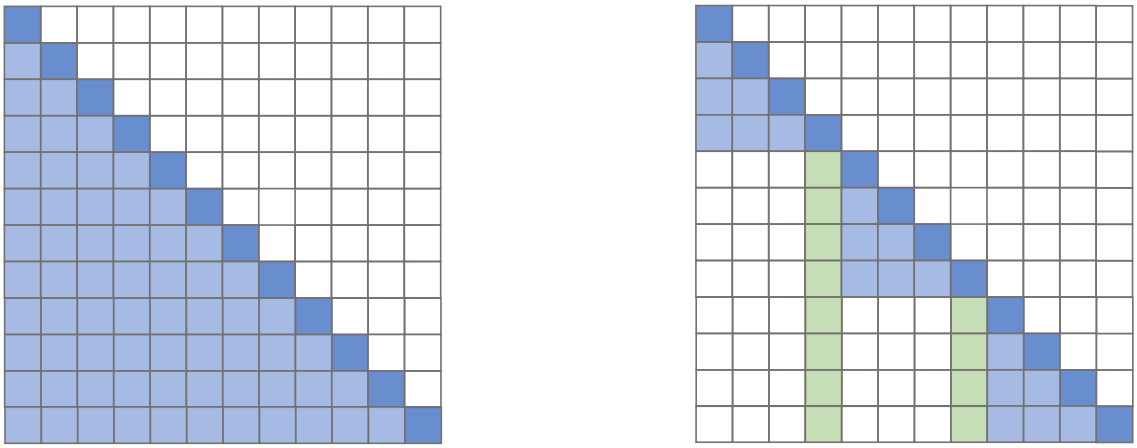
$$\begin{aligned}
\mathbf{MAB}(\mathbf{X}, \mathbf{Y}) &:= \text{LayerNorm}\left(H + \text{rFF}(H)\right), \\
H &:= \text{LayerNorm}\left(X + \text{Multihead}(X, Y, Y)\right), \\
\mathbf{SAB}(\mathbf{X}) &:= \text{MAB}(X, X), \\
\mathbf{ISAB_m}(\mathbf{X}) &:= \text{MAB}\left(X, \text{MAB}(I_m, X)\right). \\
\mathbf{PMA_k}(\mathbf{X}) &:= \text{MAB}\left(S_k, \text{rFF}(X)\right).
\end{aligned}$$

Here, $X \in \mathbb{R}^{N \times d}$ represents N, d-dimensional input/outputs stacked row-wise and rFF is a parameterized feed-forward layer that operates on each row of its input matrix separately. $I_m \in \mathbb{R}^{m \times d}$ represents m *trainable* d-dimensional "inducing points" while $S_k \in \mathbb{R}^{k \times d}$ represent k trainable d-dimensional "seed vectors" (with k set to 1 except when $k > 1$ correlated outputs are needed). The Set Transformer's encoder is just N layers of either SAB or ISAB (with N often set to 2 in practice) while its decoder is given by:

$$\mathbf{Decoder}(\mathbf{X}) := \text{rFF}\left(\text{SAB}\left(\text{PMA}_k(X)\right)\right).$$

It is straightforward to see that both ISAB and SAB are *permutation equivariant* - in other words, if the input is permuted in some way then the corresponding output of the block is permuted in exactly the same way. Meanwhile, the pooling layer PMA is permutation invariant. Since functional composition, i.e. layering, preserves these properties, the Set Transformer encoder-decoder combination is permutation invariant.

Efficiency We can understand the m inducing points I_m learned in each ISAB layer, as a form of static memory. In addition to reducing the $\mathcal{O}(Nn^2)$ complexity of the self-attending SAB layer to $\mathcal{O}(Nmn)$, a reduction particularly valuable when the input set is large, the inducing points effectively encode some global structure that helps explain its inputs. For example, in the problem of *amortized clustering*, where one attempts to learn to map an



(a) Transformer (b) Sparse Transformer

Figure 4: Illustration of patterns of the attention matrix for dense self-attention in Transformers and sparse fixed attention in Sparse Transformers.

input set of points to the centers of clusters of points inside the set, the inducing points learned could be appropriately distributed so that the encoder can effectively compare query elements with each other implicitly via their proximity to the inducing points.

The trainable k seeds S_k used in the pooling layer PMA_k can be viewed as static memory in a similar light, reducing the memory and runtime complexity of the architecture.

3.2.4 Sparse Transformer

The Sparse Transformer (Child et al., 2019) presents a simple initial attempt to reduce the quadratic complexity of the standard self-attention mechanism. The key idea is to reduce the dense attention matrix to a sparse version by only computing attention on a sparse number of q_i, k_j pairs. Sparse Transformer employs fixed attention patterns which are defined by strides and local neighbourhoods. Computation is *factorized*, wherein local and stride patterns are split amongst the heads.

Local Attention Heads Half of the heads in the Sparse Transformer are dedicated to local attention.

$$\hat{A}_{ij} = \begin{cases} Q_i(K)_j^\top), & \text{if } \lfloor j/N \rfloor = \lfloor i/N \rfloor \\ 0 & \text{otherwise} \end{cases}$$

where A_{ij} is the attention weight of q_i, k_j and $\lfloor \, \rfloor$ denote the floor operation. In this case, we only compute the attention if $\lfloor j/N \rfloor = \lfloor i/N \rfloor$ (within the same block).

Strided Attention Heads The other half of the heads are dedicated to fixed strided patterns. This is also referred to as strides. Concretely,

$$\hat{A}_{ij} = \begin{cases} Q_i(K)_j^\top), & \text{if } (i-j) \mod N = 0 \\ 0 & \text{otherwise} \end{cases}$$

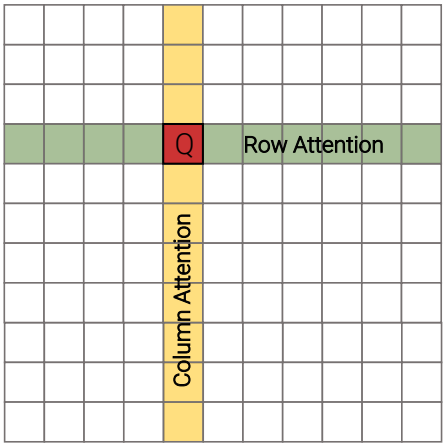


Figure 5: Attention span in Axial Transformer on a two-dimensional input.

The final result of the factorized sparse attention is visualized in Figure 4.

Parameter and Memory Complexity The modification in the self-attention mechanism does not alter the parameter costs of the model since the model still retains the Q, K, V transforms from the original Transformer model. The memory complexity of the attention layer is reduced from $\mathcal{O}(n^2)$ to $\mathcal{O}(n \log n)$.

Restrictions The Sparse Transformer implementation requires custom GPU kernels to implement a specific block-sparse variant of matrix-matrix-multiplication and cannot be easily implemented on other hardware such as TPUs.

3.2.5 Axial Transformer

Axial Transformer (Ho et al., 2019) uses factorization in a simple yet effective setup for the self-attention mechanism to process large inputs that are organized as multidimensional tensors. Instead of applying attention to the flattened version of the input, Axial Transformer simply applies multiple attentions, each along a single axis of the input tensor. Each attention, in fact, mixes information along a particular axis, while keeping information along other axes independent. Since the length of any single axis is typically much smaller than the total number of elements, Axial Transformer significantly saves computation and memory.

Axial Transformer offers an encoder-decoder architecture. For the decoding, to be able to implement the casual mask, Axial Transformer combines axial attentions with shift operations. For instance, for a model on 2-dimensional tensors, pixels are generated in raster order and to do that, first, the model encodes all pixels through an unmasked row and unmasked column attention. Then, for each row, the model applies an unmasked row and masked column attention to integrate the previously sampled rows. Finally, the model shifts the encoded representation up to make sure the conditioning information satisfies causality, and runs a masked row-attention to sample a new row in the image.

An advantage of Axial Transformer over similar methods like Sparse Transformer is that while it provides the global receptive field, it is straightforward to implement it and does not require a custom kernel for an efficient implementation.

3.2.6 Longformer

Longformer (Beltagy et al., 2020) is a variant of Sparse Transformer. Its key distinction compared to Sparse Transformer is "Dilated Sliding Windows", which can enable better long-range coverage without sacrificing sparsity. This is achieved by increasing the receptive fields by having gaps in the attention patterns. The Longformer also gradually increases the receptive field as the model goes deeper, dedicating lower levels for modeling local patterns and upper levels for modeling global patterns.

Global Attention For classification tasks, the Longformer adopts global tokens (e.g., CLS tokens) that have access to all input sequences.

Parameter and Memory Complexity The complexity of the model is reduced from $\mathcal{O}(n^2)$ to $\mathcal{O}(nk)$ where k is the size of the window. When using global attention, the Longformer creates another set of query-key-value projections for this global attention, doubling the cost of the parameters at the attention layer.

3.2.7 Extended Transformer Construction (ETC)

The ETC model (Ainslie et al., 2020) is another variation in the Sparse Transformer family. It introduces a new global-local attention mechanism. There are four components to this new attention mechanism, namely (1) global-to-global (g2g), global-to-local (g2l), local-to-global (l2g) and local-to-local (l2l).

Aside from the original input to the model, ETC introduces n_g auxiliary tokens as a prefix to the original input sequence. These tokens are regarded as global tokens and take part in global-to-$*$ and $*$-to-global attention. The local-to-local component acts as the local attention with a fixed radius of k. Overall, ETC is quite similar to the Longformer in the way it introduces global auxiliary tokens. These tokens are trainable parameters and can be interpreted as a form of memory that pools across the sequence to collect global sequence information.

Memory and Parameter Complexity The memory complexity of the ETC model is $\mathcal{O}(n_g^2 + n_g N)$, where n_g is the number of global tokens and N is the input sequence length.

Restrictions Intuitively, it is easy to observe that ETC cannot be used for auto-regressive decoding. This is because we are not able to compute causal masks because of the global attention.

3.2.8 BigBird

The BigBird model (Zaheer et al., 2020) is another Transformer for modeling longer sequences and is primarily built on top of ETC (Ainslie et al., 2020). The Big Bird model comprises of several key components, namely (1) global tokens, (2) random attention (queries attend to random keys) and (3) fixed patterns (local sliding windows).

Global Attention Fundamentally, the idea of using global memory can be traced all the way back to Longformer/ETC and Set Transformer model. Notably, the global memory in Big Bird is extended to contain tokens within the sequence, instead of simply parameterized memory. The authors call this the *'internal transformer construction (ITC)'* in which a subset of indices is selected as global tokens. This can be interpreted as a memory based approach.

Sliding Window Attention The window-ed attention was first proposed in early local-based attention models (Image Transformer, Compressed Attention and/or Sparse Transformer). In BigBird, each query attends to $w/2$ tokens to the left and $w/2$ tokens to the right. This corresponds to a fixed pattern (FP) approach.

Random Attention Finally, each query attends to r random keys. This pattern is fixed.

Memory and Parameter Complexity The memory complexity of the self-attention is linear, i.e., $O(n)$. The BigBird model does not introduce new parameters beyond the Transformer model.

Restrictions Similar to ETC, the BigBird model cannot be used to autoregressively decode. Hence, qualifying it as an encoder-only model.

3.2.9 Routing Transformer

The Routing Transformer (Roy et al., 2020) is a content-based sparse attention mechanism. It proposes a clustering-based attention mechanism that learns the attention sparsity in a data driven fashion. The first step is to project Q and K into a routing matrix R of dimensions $n \times d$.

$$R = QW_R + KW_R \tag{1}$$

where W_R is a $d \times d$ orthonormal projection matrix.

k-means Clustering The R matrix undergoes k-means clustering with a series of parameterized cluster centroids $u_1, u_2 \cdots c_k$. The k-means in Routing Transformer is trained in an online fashion. To ensure a similar number of tokens in a cluster, the model initializes $\sqrt{n}$ clusters, computes each token's distance against the cluster centroid, and takes an equal top-k for each centroid. Since the cluster centroids are trainable parameters, this is also reminiscent of the *all-attention* layer proposed by (Sukhbaatar et al., 2019).

Routing Strategy Thereafter, the routing strategy is defined as:

$$X_i' = \sum_{j \in C_i, j \leq i} A_{ij} V_j \tag{2}$$

where C_i is the cluster that vector R_i is assigned to. In other words, the token at i only attends to tokens in the same cluster.

Memory and Parameter Complexity The Routing Transformer introduces additional parameters in the clustering mechanism, namely a $k \times d$ centroid vectors and a W_r projection matrix. The memory complexity is $\mathcal{O}(n^{1.5})$.

3.2.10 Reformer

Reformer (Kitaev et al., 2020) is another efficient attention model based on locality sensitive hashing (LSH). The Reformer also introduces *reversible* Transformer layers, which contributes to further reducing its memory footprint.

LSH Attention The LSH attention introduces parameter-sharing between query and keys. It hashes the query-keys into buckets using a random-projection based hashing function. The key idea is that nearby vectors should obtain a similar hash while distant vectors should not, hence being termed as *'locality sensitive'*. To perform hashing, a random matrix $R \in \mathbb{R}^{k \times b/2}$ is first introduced. Next, The hashing function is defined as:

$$h(x) = \arg\max([xR; -xR]) \tag{3}$$

where $[;]$ is the concatenation of two vectors. For all queries, attention is computed if and only if the query and key hashes match, i.e., $h(q_i) = h(k_j)$. In other words, attention is computed amongst query and keys if they fall in the same hash bucket. In order to maintain causal masking (the ability to auto-regressively decode), the Reformer assigns and maintains a position index for every query/key. It is therefore able to compare if each query key comparison is auto-regressively valid.

Memory Efficiency with LSH Attention The key idea behind LSH attention is to classify tokens into buckets and then process them bucket by bucket in a chunked fashion. To this end, queries are first sorted by bucket number and then by sequence order within the same bucket. During computation, tokens only attend to same bucket in its own chunk and previous chunk. The chunking and sorted bucketing techniques help to improve the overall efficiency of the Reformer model.

Parameter and Memory Complexity The memory complexity of the Reformer is $\mathcal{O}(n \log n)$. In terms of parameter costs, the Reformer shares queries and keys, which reduces the cost of the QKV transforms by a third. The random projections are not trainable parameters and hence do not incur parameter costs. Overall, the Reformer has fewer parameters than vanilla Transformers. The reversible layers in Reformer also reduce the memory consumption during training by enabling activations to be reconstructed from the next layer's. This reduces memory cost since this eliminates the need to store activations for all layers during backpropagation.

3.2.11 Sinkhorn Transformers

This section introduces the Sparse Sinkhorn Transformer (Tay et al., 2020b). The Sinkhorn Transformer belongs to the family of *learned patterns*. This model is a chunked/blocked model that learns sparse patterns by re-sorting the input key and values in a block-wise fashion and then applying local block-based attention.

$$A_{ij} = \begin{cases} (Q_i \psi_S(K)_j^\top), & \text{if} \lfloor j/N \rfloor = \lfloor i/N \rfloor \\ 0 & \text{otherwise} \end{cases}$$

where ψ_S applies a sorting operator on the sequence length dimension.

Sorting Network The sorting operator is parameterized by a meta sorting network. Let X be the input sequence of dimension $N \times d$.

$$\psi_S(X) = \phi_S(F_S(\textsc{BlockSum}(X)))\ \textsc{BlockShape}(X) \tag{4}$$

where $F_S(.)$ is a parameterized function such as a two layer feed-forward network with ReLU activation. The output of $F_S(.)$ is a tensor of $n_B \times n_B$. The BlockSum function learns the sum embeddings of local blocks. The BlockShape function reshapes the input tensor into $\mathbb{R}^{N \times d} \rightarrow \mathbb{R}^{n_B \times b \times d}$. Here, we note that $N = n_B \times b$, where b is the size of the block and n_B is the number of total blocks.

Sinkhorn Sorting ϕ is the Sinkhorn balancing operator (Sinkhorn, 1964; Adams and Zemel, 2011) which converts the $n_B \times n_B$ matrix into a soft permutation matrix. Specifically, a series of row- and column-wise normalizations are applied on the matrix output of $F_S\text{BlockSum}(X)$. For the sake of brevity, we do not delve into details of this operation. Further details can be found at (Adams and Zemel, 2011; Tay et al., 2020b).

Parameter and Memory Complexity The memory complexity of the Sinkhorn Transformer is $\mathcal{O}(b^2)$ where b is the block size and $b = \frac{N}{N_b}$. Additional parameter costs are incurred from the meta sorting network $F_S(.)$. The number of additional parameters is therefore $2d^2$ when a two layer ReLU network is used as the sorting network.

3.2.12 Linformer

The Linformer (Wang et al., 2020b) is an efficient Transformer based on the idea of low-rank self-attention.

Low Rank Projections on Length Dimensions The Linformer projects the $N \times d$ dimensional keys and values to $k \times d$ dimensions using additional projection layers. Note that this is a reduction on the length dimension instead of the key and value dimensions. Given the newly projected keys (K') and values (V'), the QK' matrix is now $(N \times k)$ dimensions instead of $(N \times N)$. The attention matrix $\text{Softmax}(QK')$ multiplies with $V' \in \mathbb{R}^{k \times d}$ to result in an output tensor of dimensions $N \times d$. To some extent, Linformer is reminiscent of depth-wise convolutions (Kaiser et al., 2017). A projection on the length dimension causes mixing of sequence information (dimension-wise) in a single transformation. Hence, it is non-trivial to maintain causal masking and/or prevent mixing of past and future information when computing attention scores.

Parameter and Memory Complexity The memory complexity of Linformer is $\mathcal{O}(n)$. There is only a minimal parameter costs of the Linformer due to the extra $N \times k$ length projections. If k is sufficiently small, there is negligible parameter costs incurred.

3.2.13 Linear Transformer

The Linear Transformer (Katharopoulos et al., 2020) improves the complexity of self-attention from quadratic to linear by using a kernel-based formulation of self-attention and the associative property of matrix products. Furthermore, it reduces attention with casual masking (which is used in auto-regressive decoding) to a linear-time, constant memory

recurrent neural network (RNN). The model has been shown to improve inference speeds up to *three orders of magnitude* without much loss in predictive performance.

The method rests on the simple but powerful observation that the accumulated value V_i' for the query Q_i in position i can be written as:

$$V_i' = \frac{\sum_{j=1}^{p} \text{sim}(Q_i, K_j)V_j}{\sum_{j=1}^{p} \text{sim}(Q_i, K_j)}.$$

Here, $p = N$ in full, unmasked attention and $p = i$ in the case of causal masking. Now, in usual softmax attention, $\text{sim}(q, k) = \exp\left(\frac{q^T k}{\sqrt{d}}\right)$. Linear Transformer, however, expresses the similarity as a kernel function. That is, $\text{sim}(q, k) := \phi(q)^T \phi(k)$, where ϕ is a, possibly high-dimensional, feature map. With this choice, we can rewrite V_i' as:

$$\begin{aligned} V_i' &= \frac{\phi(Q_i)^T S_p}{\phi(Q_i)^T Z_p}, \\ S_p &:= \sum_{j=1}^{p} \phi(K_j)V_j^T, \\ Z_p &:= \sum_{j=1}^{p} \phi(K_j). \end{aligned}$$

For unmasked attention, since $p = N$ we only need to compute S_N and Z_N once and we reuse them for the computation at every position $0 \leq i \leq N$. For causal attention, the S_i's and Z_i's can be viewed as states of an RNN that are updated by the following recurrence relations:

$$\begin{aligned} S_i &= S_{i-1} + \phi(K_i)V_i^T, \\ Z_i &= Z_{i-1} + \phi(K_i) \end{aligned}$$

with initial condition $S_0 = Z_0 = 0$. If the dimension of the key, query, and values are all d and cost to compute ϕ is $\mathcal{O}(c)$, then the overall run-time complexity of Linear Transformer is $\mathcal{O}(Ncd)$. The authors choose

$$\phi(x) = \text{elu}(x) + 1,$$

where $\text{elu}(\cdot)$ denotes the exponential linear unit (Clevert et al., 2015). With this choice of feature map, $c = d$ and the end-to-end complexity of the model is $\mathcal{O}(Nd^2)$. The authors go further and show that in addition to the forward pass, the *backward* pass (i.e. gradient computation) can be achieved in linear time and constant memory by using cumulative sums. We defer readers interested in this derivation to the original paper.

3.2.14 Performer

The Performer (Choromanski et al., 2020) model is characterized by its Generalized Attention mechanism and its usage of random Kernels.

Generalized Attention The generalized attention entangles Q_i, K_j with a kernel function K. The attention matrix in Performer is computed via:

$$A = [g(Q_i^\top)K(Q_i^\top K_j^\top), h(K_j^\top)] \tag{5}$$

where $K(.)$ is a kernel function that maps $d \times d$ to a scalar value $\mathbb{R}$ and g, h are functions that map d to a scalar value $\mathbb{R}$.

Fast Attention via Orthogonal Random Features (FAVOR) The above computation is still quadratic in complexity. Hence, the Performer leverages approximation tricks to avoid storing and computing the $N \times N$ attention matrix. It leverages *orthogonal random features* (ORF) for doing so. The final attention output Y of the Performer is described as follows:

$$Y = \hat{D}^{-1}(Q'((K')^\top V)) \tag{6}$$

where $\hat{D} = \text{diag}(Q'((K')^\top 1_N))$, $Q' = D_Q\phi(Q^\top)^\top$, and $K' = D_K\phi(K^\top)^\top$. Note that $D_Q = g(Q_i^\top), D_K = h(K_i^\top)$. The function $\phi(x)$ is defined as:

$$\phi(X) = \frac{c}{\sqrt{M}} f(Wx + b)^\top \tag{7}$$

where $c > 0$ is a constant, $W \in \mathbb{R}^{M \times d}$ is a random feature matrix, and M is the dimensionality of this matrix that controls the number of random features. We are able to see that we do not explicitly compute $A = QK^\top$ and hence avoid paying the N^2 cost. For rigorous theoretical analysis and further details, we refer interested readers to (Choromanski et al., 2020).

Parameter and Memory Complexity The complexity of bidirectional FAVOR algorithm is $\mathcal{O}(Md + Nd + MN)$ where M is the dimensionality of the random features.

3.2.15 Synthesizers

Synthesizer models (Tay et al., 2020a) are an attempt to study and investigate the true importance of conditioning within the self-attention mechanism. In (Tay et al., 2020a), the authors study a synthetic self-attention module in which attention weights are approximated instead of being computed by pairwise dot products. Synthesizers are only implicitly related to efficient Transformers. However, the factorized variants can be considered a low-rank efficient Transformer model.

Dense Synthesizers In the Dense Synthesizer, each token x_i is projected to a vector of length N using a two-layered non-linear feed-forward network. The computation of the attention matrix A is described as:

$$A = W_2(\sigma_R(W_1(X) + b)) + b \tag{8}$$

where $X \in \mathbb{R}^{N \times d}$ is the input sequence, $W_2 \in \mathbb{R}^{d \times N}, W_1 \in \mathbb{R}^{d \times d}$, and σ_R is the ReLU activation function. Given A, the output of the Synthetic Dense function is computed as:

$$Y = \text{Softmax}(A)G(X). \tag{9}$$

where $G(X)$ is another parameterized function $\mathbb{R}^{N \times d} \to \mathbb{R}^{N \times d}$.

Random Synthesizers Another variant of the Synthesizer model uses random matrices for A. In this case, the output can be expressed by:

$$Y = \text{Softmax}(R)G(X). \tag{10}$$

where $R \in \mathbb{R}^{N \times N}$ is a trainable and/or non-trainable matrix. In (Tay et al., 2020a), the authors show that Random Synthesizers achieve competitive performance.

Factorized Variants The Dense and Random Synthesizers also come with factorized variants that consider a low-rank structure of the attention matrix. For factorized random Synthesizer can be written as:

$$Y = \text{Softmax}(R_1 R_2^\top)G(X). \tag{11}$$

where $R_1, R_2 \in \mathbb{R}^{N \times k}$. On the other hand, the Dense Synthesizer can be factorized as follows:

$$A = H_B(B) * H_C(C) \text{ where } B, C = F_B(X_i), F_C(X_i), \tag{12}$$

where $F_B(.)$ projects onto b dimensions and $F_C(.)$ projects X_i onto c dimensions with $c \times b = N$. H_B, H_C are tile and repeat functions respectively.

Parameter and Memory Complexity For Random Synthesizers that adopt a non-trainable R, there is no need to store N^2 activations at this layer. For the trainable Random Synthesizer, the memory complexity and parameter complexity remains as N^2. However, there is no need to compute N^2 dot products, reducing the computational costs significantly. The Factorized Random Synthesizers reduce the parameter costs to $2(N \times k)$.

3.2.16 Transformer-XL

The Transformer-XL model (Dai et al., 2019) relies on segment-based recurrence. Segment-based recurrence can be considered an orthogonal approach to the other techniques discussed since it does not explicitly sparsify the dense self-attention matrix. Instead, it connects adjacent blocks with a recurrent mechanism.

Segment Recurrence The recurrent mechanism in Transformer-XL is described as:

$$\tilde{\boldsymbol{h}}_{\tau+1}^{n-1} = [\text{SG}(\boldsymbol{h}_{\tau}^{n-1}) \odot \boldsymbol{h}_{\tau+1}^{n-1}] \tag{13}$$

$$q_{\tau+1}^{n}, k_{\tau+1}^{n}, v_{\tau+1}^{n} = \boldsymbol{h}_{\tau+1}^{n-1}\boldsymbol{W}_q^\top \ , \ \tilde{\boldsymbol{h}}_{\tau+1}^{n-1}\boldsymbol{W}_k^\top \ , \ \tilde{\boldsymbol{h}}_{\tau+1}^{n-1}\boldsymbol{W}_v^\top \tag{14}$$

$$\boldsymbol{h}_{\tau+1}^{n} = \text{Transformer}(q_{\tau+1}^{n}, k_{\tau+1}^{n}, v_{\tau+1}^{n}) \tag{15}$$

where SG() is the stop gradient function, $\odot$ is the concatenation of two sequences along the length dimension. Notably, the keys and values are conditioned on the previous sequence length $\tilde{\boldsymbol{h}}_{\tau+1}^{n-1}$ instead of $\boldsymbol{h}_{\tau+1}^{n-1}$

Relative Positional Encodings Transformer-XL introduces novel relative position encodings. In this scheme, absolute positional encodings are not added to the content embeddings. Instead, they are only considered while computing attention weights where they can be replaced with relative position encodings. Since the relative position encodings are not directly relevant to the efficiency of the model, we refer interested readers to (Dai et al., 2019) for more details.

3.2.17 Compressive Transformers

Compressive Transformers (Rae et al., 2020) are a natural extension of the Transformer-XL model. The key idea behind the Compressive Transformer is to maintain a fine-grained memory of past segment activations. This is unlike Transformer-XL, which discards past activations as it moves across segments.

Memory The Compressive Transformer is characterized by a dual memory system - a primary memory and a secondary compressed memory. It maintains a memory with n_m memory slots and n_{cm} compressive memory slots. Whenever the model accepts a new input segment, the oldest n_s activations in the primary memory are moved to the compressed memory where a compression function is applied.

Compression These memories are compressed with a variety of compression functions such as (1) mean/max pooling (2) 1D convolutions, (3) dilated convolutions, and (4) most used (e.g., sorted by usage of attention).

Memory Reconstruction In order to better retain memories over long sequences, the Compressive Transformer implements an auto-encoding loss that learns to reconstruct the original memory from its compressed version, i.e., $L^{ae} = ||\text{old\_mem} - g(\text{new\_cm}^{(i)})||$ where $g(.) : \mathbb{R}^{\frac{n_s}{c} \times d} \rightarrow \mathbb{R}^{n_s \times d}$ is a parameterized function. A second attention reconstruction is a lossy re-construct that attempts to reconstruct the attention over memory instead of the lossless reconstruction of the memory itself.

Computational and Memory Complexity In terms of memory and computational complexity, on a square image of size N, Axial Transformer performs the attention computation in $\mathcal{O}(n\sqrt{n})$, which saves $\mathcal{O}(\sqrt{n})$ over normal self-attention. For instance, with on square image with N pixels, organized in a $b \times b$ grid, Axial Transformer runs b attention sequences of length b, which is of complexity $\mathcal{O}(b.b^2)$. In a more general case, for a d-dimensional tensor of shape $N = N^{1/d} \times \ldots \times N^{1/d}$, Axial Transformer saves a $\mathcal{O}(N^{(d-1)/d})$ factor of resources over standard self-attention.

4. Discussion

This section the state of research pertaining to this class of efficient models.

4.1 On Evaluation

While the field is bustling with new Transformer models, there is hardly an easy way to compare these models side by side. Many research papers select their own benchmarks to showcase the abilities of the proposed model. This is also coupled with different hyperparameter settings like model sizes and configurations which can make it difficult to correctly attribute the reason for the performance gains Moreover, some papers conflate this with pretraining (Devlin et al., 2018) which makes it even harder to distinguish the relative performance of these different models. It is still a mystery to which fundamental efficient Transformer block one should consider using.

On one hand, there are multiple models that focus on generative modeling, showcasing the ability of the proposed Transformer unit on auto-regressive modeling of sequences. To

this end, Sparse Transformers (Child et al., 2019), Adaptive Transformers (Correia et al., 2019), Routing Transformers (Roy et al., 2020) and Reformers (Kitaev et al., 2020) are mainly focused on generative modeling tasks. These benchmarks typically involve language modeling and/or pixel-wise image generation on datasets such as wikitext, enwik8 and/or ImageNet/CIFAR. Models that use segment based recurrence such as Transformer-XL and Compressive Transformers are also focused on long-range language modeling tasks such as PG-19.

On one hand, a collection of models is mainly focused on encoding-only tasks such as question answering, reading comprehension and or selections from the Glue benchmark. For example, the ETC model (Ainslie et al., 2020) only runs experiments on question answering benchmarks such as NaturalQuestions or TriviaQA. On the other hand, the Linformer (Wang et al., 2020b) focuses on subsets of the GLUE benchmark. This split is very natural and intuitive, since models like ETC and Linformer cannot be used in an auto-regressive fashion, i.e., cannot be used to decode. This aggravates the difficulty of comparing these encoder-only models with the other models.

There are models that focus on a balance of both. The Longformer (Beltagy et al., 2020) tries to balance this by running benchmarks on both generative modeling and encoder-only tasks. The Sinkhorn Transformer (Tay et al., 2020b) compares on both generative modeling tasks as well as encoding only tasks.

Additionally, it is also worthy to note that, although Seq2Seq machine translation (MT) was one of the problems that popularized Transformer models, not many of these efficient Transformer models evaluate on MT. This is likely because sequence lengths in MT is not long enough to warrant the usage of these models.

While generative modeling, glue tasks and/or question answering appear to be the common evaluation benchmarks adopted by many of these tasks, there are several niche benchmarks that a small isolated number of papers choose to evaluate on. For starters, the Performer model (Choromanski et al., 2020) evaluates on masked language modeling on proteins, deviating from serious head-on comparisons with other efficient Transformer models. The Linear Transformer (Katharopoulos et al., 2020) also evaluates on speech recognition, which is a rare benchmark amongst this group of papers.

4.2 On Model Design Trends

When matching our broad categorization against the timeline of the introduction of these models, we are able to see the trend that the community is taking towards designing efficient Transformer models. Early work in this area has primary been dominated by more intuitive and simple approaches such as *fixed patterns*. To this end, most early work in this area seems to be based on block/local patterns such as Image Transformer (Parmar et al., 2018), Compressed Attention (Liu et al., 2018), Blockwise Transformer (Qiu et al., 2019) or the local windows in Sparse Transformer (Child et al., 2019).

The paradigm of factorizing various fixed patterns was first introduced in (Child et al., 2019) and the Axial Transformer (Ho et al., 2019). At this particular same time, we start to observe early traces of *memory* based approaches from both the inducing point methods in the Set Transformer (Lee et al., 2019) or global nodes in the Star Transformer (Guo et al., 2019a) model.

We observe the next wave of models comes in the form of learnable sparsity patterns. Reformer (Kitaev et al., 2020) and Routing Transformers (Roy et al., 2020) are very similar in the sense that they are models that learn to cluster/bucket tokens before performing attention. The key difference is the means to the end whereby Reformer uses a hashing function while the Routing Transformer uses online k-means for cluster assignment. In parallel, Sinkhorn Transformers (Tay et al., 2020b) are also based on the idea of sorting, albeit at block-level. These three models largely follow a similar paradigm of re-arranging sequences for efficient computation of attention scores.

Next, we then observe several extensions that are largely built off the Sparse Transformer paradigm. The ETC (Ainslie et al., 2020) and Longformer (Beltagy et al., 2020) models are very similar ideas that are fundamentally Sparse Transformer extensions. These models incorporate the notion of a global memory, which is reminiscent of the Set Transformer's inducing point method or the global memory of the Star Transformer. Modifications to strides, such as using dilated windows was also proposed in the Longformer work.

The most recent wave of models we've been seeing is models that are based on low-rank approximation or kernel methods, e.g., models such as Linformer (Wang et al., 2020b), Performer (Choromanski et al., 2020) and/or Linear Transformers (Katharopoulos et al., 2020). Although due to the state of evaluation and the high parallelism of research, it is quite unclear if this low-rank or kernel paradigm is actually better than the learnable pattern (LP) or memory based efficient Transformer models.

On the side, it is good to note that the recurrent based models (Transformer-XL and Compressive Transformers) seem to operate orthogonally and are in less direct comparison with the other models.

4.3 Brief Discussion on Orthogonal Efficiency Efforts

While this paper is focused on the computational and memory complexity of the self-attention module, we briefly summarize several orthogonal efforts that may also contribute to model efficiency, scalability and overall usability of Transformer models.

- **Weight Sharing** Sharing parameters of the Transformer models would help in reducing overall model size. The Universal Transformers (Dehghani et al., 2018) tie attention and transition weights across layers. Similarly, Albert (Lan et al., 2019) does the same parameter sharing across layers. On the other hand, the Quaternion Transformer (Tay et al., 2019) proposes a weight sharing scheme inspired by Hamilton products that locally shares the components in the linear transformation layers.

- **Quantization / Mixed Precision** Learning mixed precision models has the potential to improve memory costs. The Q-BERT (Shen et al., 2020) is a model that quantizes Transformer models to ultra-low precision. Meanwhile mixed precision training (Ott et al., 2019) is a highly popular technique to reduced the memory costs of training Transformers. (Fan et al., 2020) applies Quantization Aware training to Transformer models.

- **Knowledge Distillation** Knowledge distillation (KD) (Hinton et al., 2015) has been a useful technique for transfering the knowledge learned from a larger teacher model to a smaller student model. The smaller model can then be efficiently deployed into

production. There have been many attempts to distill large Transformer models. For example, DistilBERT (Sanh et al., 2019), task-specific distillation (Tang et al., 2019) and TinyBERT (Jiao et al., 2019).

- **Neural Architecture Search (NAS)** Searching for more efficient Transformer architectures is also a common strategy. (Guo et al., 2019b) proposed Neural Architecture Transformer (NAT), using NAS to search for more compact and efficient Transformers by removing redundant operations. (Wang et al., 2020a) proposed HAT (Hardware-aware Transformers), a method that leverages NAS and uses hardware efficiency feedback as a reward signal.

- **Task Adapters** This line of research has been primarily focused on the problem of fine-tuning large Transformer on T tasks and aiming to reuse parameters across a variety of tasks. The key idea is that task adapters (Houlsby et al., 2019) enable reuse of parameters across tasks and reuse the need of serving T models in production - resulting in overall parameter savings. A modest number of models have been proposed, such as PALS (Stickland and Murray, 2019), MAD-X (Pfeiffer et al., 2020) and HyperGrid (Tay et al., 2020c).

5. Conclusion

In this paper we surveyed the literature on efficient Transformer models especially pertaining ot the quadratic complexity of the self-attention module. We provided a taxonomy and high-level abstraction of the core techniques employed in these class of new models. We characterize the existing models based on techniques and provided a comprehensive walkthrough on several of the efficient Transformer models. Finally, we discussed the evaluation landscape of these models along with the design trends of these models. We ended of with a brief discussion of other parallel orthogonal efforts that may improve the efficiency of Transformer models in general.

References

Ryan Prescott Adams and Richard S Zemel. Ranking via sinkhorn propagation. *arXiv preprint arXiv:1106.1925*, 2011.

Karim Ahmed, Nitish Shirish Keskar, and Richard Socher. Weighted transformer network for machine translation. *arXiv preprint arXiv:1711.02132*, 2017.

Joshua Ainslie, Santiago Ontanon, Chris Alberti, Philip Pham, Anirudh Ravula, and Sumit Sanghai. Etc: Encoding long and structured data in transformers. *arXiv preprint arXiv:2004.08483*, 2020.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

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

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. *arXiv preprint arXiv:2005.12872*, 2020.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Jared Davis, Tamas Sarlos, David Belanger, Lucy Colwell, and Adrian Weller. Masked language modeling for proteins via linearly scalable long-context transformers. *arXiv preprint arXiv:2006.03555*, 2020.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

Gonçalo M Correia, Vlad Niculae, and André FT Martins. Adaptively sparse transformers. *arXiv preprint arXiv:1909.00015*, 2019.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. *arXiv preprint arXiv:1807.03819*, 2018.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Angela Fan, Pierre Stock, Benjamin Graham, Edouard Grave, Remi Gribonval, Herve Jegou, and Armand Joulin. Training with quantization noise for extreme fixed-point compression. *arXiv preprint arXiv:2004.07320*, 2020.

Qipeng Guo, Xipeng Qiu, Pengfei Liu, Yunfan Shao, Xiangyang Xue, and Zheng Zhang. Star-transformer. *arXiv preprint arXiv:1902.09113*, 2019a.

Yong Guo, Yin Zheng, Mingkui Tan, Qi Chen, Jian Chen, Peilin Zhao, and Junzhou Huang. Nat: Neural architecture transformer for accurate and compact architectures. In *Advances in Neural Information Processing Systems*, pages 737–748, 2019b.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Jonathan Ho, Nal Kalchbrenner, Dirk Weissenborn, and Tim Salimans. Axial attention in multidimensional transformers. *arXiv preprint arXiv:1912.12180*, 2019.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. *arXiv preprint arXiv:1902.00751*, 2019.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. Tinybert: Distilling bert for natural language understanding. *arXiv preprint arXiv:1909.10351*, 2019.

Lukasz Kaiser, Aidan N Gomez, and Francois Chollet. Depthwise separable convolutions for neural machine translation. *arXiv preprint arXiv:1706.03059*, 2017.

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

Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=rkgNKkHtvB`.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.

Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International Conference on Machine Learning*, pages 3744–3753, 2019.

Peter J Liu, Mohammad Saleh, Etienne Pot, Ben Goodrich, Ryan Sepassi, Lukasz Kaiser, and Noam Shazeer. Generating wikipedia by summarizing long sequences. *arXiv preprint arXiv:1801.10198*, 2018.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. fairseq: A fast, extensible toolkit for sequence modeling. *arXiv preprint arXiv:1904.01038*, 2019.

Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Łukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. Image transformer. *arXiv preprint arXiv:1802.05751*, 2018.

Niki Parmar, Prajit Ramachandran, Ashish Vaswani, Irwan Bello, Anselm Levskaya, and Jon Shlens. Stand-alone self-attention in vision models. In *Advances in Neural Information Processing Systems*, pages 68–80, 2019.

Jonas Pfeiffer, Ivan Vulić, Iryna Gurevych, and Sebastian Ruder. Mad-x: An adapter-based framework for multi-task cross-lingual transfer. *arXiv preprint arXiv:2005.00052*, 2020.

Jiezhong Qiu, Hao Ma, Omer Levy, Scott Wen-tau Yih, Sinong Wang, and Jie Tang. Blockwise self-attention for long document understanding. *arXiv preprint arXiv:1911.02972*, 2019.

Jack W. Rae, Anna Potapenko, Siddhant M. Jayakumar, Chloe Hillier, and Timothy P. Lillicrap. Compressive transformers for long-range sequence modelling. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=SylKikSYDH`.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*, 2019.

Aurko Roy, Mohammad Saffar, Ashish Vaswani, and David Grangier. Efficient content-based sparse attention with routing transformers. *arXiv preprint arXiv:2003.05997*, 2020.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. Q-bert: Hessian based ultra low precision quantization of bert. 2020.

Richard Sinkhorn. A relationship between arbitrary positive matrices and doubly stochastic matrices. *The annals of mathematical statistics*, 35(2):876–879, 1964.

David R So, Chen Liang, and Quoc V Le. The evolved transformer. *arXiv preprint arXiv:1901.11117*, 2019.

Asa Cooper Stickland and Iain Murray. Bert and pals: Projected attention layers for efficient adaptation in multi-task learning. *arXiv preprint arXiv:1902.02671*, 2019.

Sainbayar Sukhbaatar, Edouard Grave, Guillaume Lample, Herve Jegou, and Armand Joulin. Augmenting self-attention with persistent memory. *arXiv preprint arXiv:1907.01470*, 2019.

Raphael Tang, Yao Lu, Linqing Liu, Lili Mou, Olga Vechtomova, and Jimmy Lin. Distilling task-specific knowledge from bert into simple neural networks. *arXiv preprint arXiv:1903.12136*, 2019.

Yi Tay, Aston Zhang, Luu Anh Tuan, Jinfeng Rao, Shuai Zhang, Shuohang Wang, Jie Fu, and Siu Cheung Hui. Lightweight and efficient neural natural language processing with quaternion networks. *arXiv preprint arXiv:1906.04393*, 2019.

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

Yi Tay, Dara Bahri, Liu Yang, Donald Metzler, and Da-Cheng Juan. Sparse sinkhorn attention. *arXiv preprint arXiv:2002.11296*, 2020b.

Yi Tay, Zhe Zhao, Dara Bahri, Donald Metzler, and Da-Cheng Juan. Hypergrid: Efficient multi-task transformers with grid-wise decomposable hyper projections. *arXiv preprint arXiv:2007.05891*, 2020c.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

Hanrui Wang, Zhanghao Wu, Zhijian Liu, Han Cai, Ligeng Zhu, Chuang Gan, and Song Han. Hat: Hardware-aware transformers for efficient natural language processing. *arXiv preprint arXiv:2005.14187*, 2020a.

Sinong Wang, Belinda Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020b.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. In *Advances in neural information processing systems*, pages 3391–3401, 2017.

Manzil Zaheer, Guru Guruganesh, Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. Big bird: Transformers for longer sequences. *arXiv preprint arXiv:2007.14062*, 2020.