Efficient Transformers: A Survey

YI TAY, Google Research, USA
MOSTAFA DEHGHANI, Google Research, Amsterdam
DARA BAHRI and DONALD METZLER, Google Research, USA

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 article 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.

CCS Concepts: • **Computing methodologies** → **Machine learning algorithms;**

Additional Key Words and Phrases: Transformers, attention, deep learning, neural networks

ACM Reference format:
Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. 2022. Efficient Transformers: A Survey. *ACM Comput. Surv.* 55, 6, Article 109 (December 2022), 28 pages.
https://doi.org/10.1145/3530811

1 INTRODUCTION

Transformers [82] 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 [6, 12, 20, 60], image processing [7, 55] and information retrieval [77]. 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 [2, 18, 68]. This immense interest has also spurred research into more efficient variants of the model [5, 11, 14, 15, 36, 37, 59, 62, 70, 74, 84, 87].

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 and this manuscript's first draft (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

Authors' addresses: Y. Tay, D. Bahri, and D. Metzler, Google Research, USA, California; M. Dehghani, Google Research, Amsterdam.

0360-0300/2022/12-ART109
https://doi.org/10.1145/3530811

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"*.

The *efficiency* of a model can be interpreted in a variety of ways. 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. Efficiency might also refer to computational costs, e.g., the number of FLOPs, both during training and inference. In particular, for on-device applications, models often must operate within a highly constrained computational budget. Throughout this survey, we refer to the efficiency of Transformers both in terms of memory and computation. We are especially interested in how such models perform when they are applied to 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 general efficiency of Transformers, including but not limited to tackling the quadratic complexity issue of the self-attention mechanism or reducing the computation costs by means such as pooling and/or sparsity. We also briefly discuss general improvements and other efficiency improvements such as parameter sharing.

We propose a taxonomy of efficient Transformer models, characterizing them by their 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 [82]. 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 [4] 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_{model} dimensional embedding, i.e., $\mathbb{R}^B \times \mathbb{R}^N \times \mathbb{R}^{d_{model}}$. 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 [82]) 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-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.

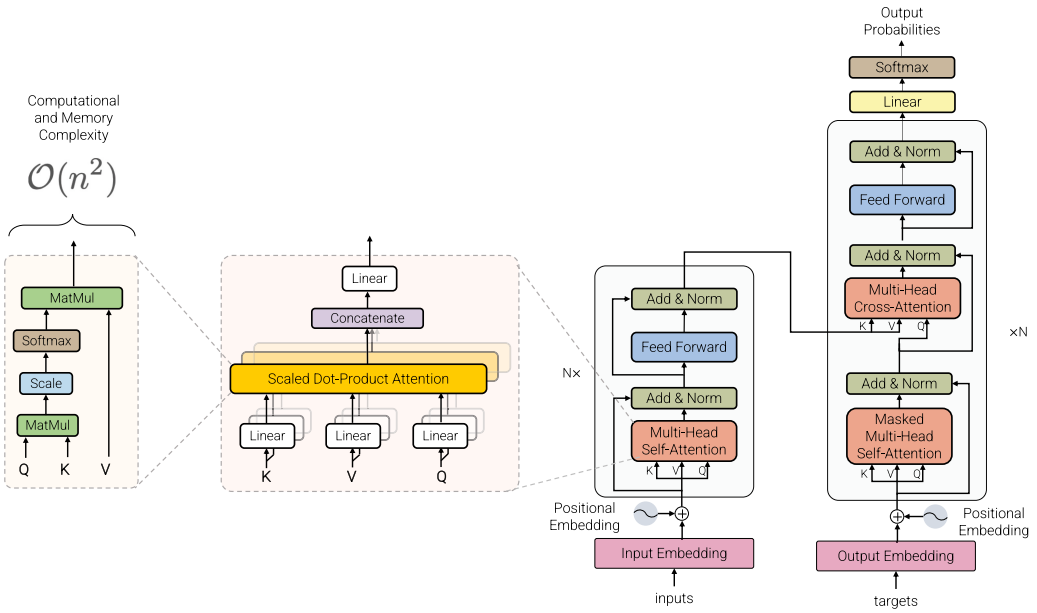


Fig. 1. Architecture of the standard Transformer [82].

2.1 Multi-Head Self-Attention

The Transformer model leverages a multi-head self-attention mechanism. The key idea behind the mechanism is for each element in the sequence to learn 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 X is a matrix in $\mathbb{R}^{N \times d}$, α is a scaling factor that is typically set to $\frac{1}{\sqrt{d}}$, $Q_h = X\boldsymbol{W}_q, K_h = X\boldsymbol{W}_k$ and $V_h = X\boldsymbol{W}_v$ are linear transformations applied on the temporal dimension of the input sequence, and $\boldsymbol{W}_q, \boldsymbol{W}_k, \boldsymbol{W}_v \in \mathbb{R}^{d \times \frac{d}{H}}$ are the weight matrices (parameters) for the query, key, and value projections that project the input X to an output tensor of d dimensions, and N_H is the number of heads. Softmax is applied row-wise.

The outputs of heads $A_1 \cdots A_H$ are concatenated together and passed into a dense layer. The output Y can thus be expressed as $Y = \boldsymbol{W}_o[A_1 \cdots A_H]$, 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 $\mathbb{R}^B \times \mathbb{R}^N \times \mathbb{R}^H \times \mathbb{R}^{\frac{d}{H}}$ 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.

2.2 Position-Wise Feed-Forward Layers

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

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

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

2.3 Putting It All Together

Each Transformer block can be expressed as:

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

where X is the input of the Transformer block and X_B is the output of the Transformer block. Note that the MultiheadAttention() function accepts two argument tensors, one for query and the other for key-values. If the first argument and second argument are the same input tensor, this is the MultiheadSelfAttention mechanism.

2.4 On the Compute Cost of Transformers

The compute costs of Transformers is derived from multiple factors. First, 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. Memory restrictions tend to be applicable more to training (due to gradient updates) and are generally of lesser impact on inference (no gradient updates). The quadratic cost of self-attention impacts speed[1] in both training and inference. The compute costs of the self-attention mechanism contributes partially to the overall compute cost of the Transformer. A non-trivial amount of compute still stems from the two-layer feed-forward layers at every Transformer block (approximately half the compute time and/or FLOPs). The complexity of the FFN is linear with respect to sequence length but is generally still costly. Hence, a large portion of recent work have explored sparsity [23, 45] as a means to scale up the FFN without incurring compute costs. Efficient attention and efficient models are generally orthogonal—although some efficient attention methods explicitly aim to reduce the sequence length [15] and as a result also save computation costs in both aspects. Efficiency and computational costs is generally a complicated affair and we would suggest readers peruse [17] for more details on trade-offs, intricacies, and the like.

2.5 Transformer Mode

It is important to note the differences in how the Transformer blocks are used. 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, self-attention used in the decoder (i.e., across decoding positions) must be causal since each auto-regressive decoding step can only depend on previous tokens, whereas the self-attention used in the encoder need not. Fulfilling this requirement can prove challenging for many efficient self-attention designs.

The mode of usage of a Transformer model generally depends on the target application. Given an input sequence, the sequence is typically passed through an encoder stack. At this stage, there might be two options. For multi-class classification, a linear layer with Softmax outputs typically

[1]We would like to emphasize that complexity does not always translate to real-world throughput or latency. A model of linear complexity can be slower than a model with quadratic complexity in practice.

projects the sequence representation down to the number of classes. In the case of BERT [20], this is a *[CLS]* token that is appended to the start of the sequence as a prefix. Recent work has also explored the usage of Encoder-Decoder architectures for classification, such as T5 [60]. Decoder-only models are typically used for generation and are trained using a language modeling objective (of predicting the next token). Due to the nature of the loss, these models are often superior for open ended generation [6]. A decoder-only model needs to be causal and a upper triangular mask needs to be applied to prevent tokens from peeping into the future. We refer interested readers to [60] for more detailed descriptions of the various Transformer modes.

2.6 Applications

Transformers have a wide range of applications ranging from language to vision, speech, and reinforcement learning. It was initially introduced within the context of sequence-to-sequence machine translation in NLP. Following which, most of the applications of Transformers have been within the context of language—given the concurrent advance of pretrained models such as BERT [20]. Many early improvements to this line of efficient transformers is therefore focused on language processing applications [3, 5]. For historical reasons, this article leans slightly towards language. However, it is also worth noting that a substantial amount of papers considered in our survey also considers multimodal applications whereby a sequence processor is required. For example, Roy et al. [62], Choromanski et al. [11], Tay et al. [74], and Child et al. [9] consider generative modeling tasks on images or other modalities such as proteins.

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. While the primary goal of most of these models is to improve the memory complexity of the self-attention mechanism, we also include methods that improve the general efficiency of the Transformer architecture.

- *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 [58] and/or Local Attention [55]. 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 [9] and/or Longformer [5] employ strided or "dilated" windows.
  - *Compressed Patterns.* Another line of attack here is to use some pooling operator to downsample the sequence length to be a form of fixed pattern. For instance, Compressed Attention [48] uses strided convolution to effectively reduce the sequence length.

Table 1. Summary of Efficient Transformer Models

Model/Article	Complexity	Decode	Class
Memory Compressed† [48]	$O(N_c^2)$	✓	FP+M
Image Transformer† [55]	$O(N.m)$	✓	FP
Set Transformer† [43]	$O(kN)$	×	M
Transformer-XL† [16]	$O(N^2)$	✓	RC
Sparse Transformer [9]	$O(N\sqrt{N})$	✓	FP
Reformer† [37]	$O(N \log N)$	✓	LP
Routing Transformer [62]	$O(N\sqrt{N})$	✓	LP
Axial Transformer [28]	$O(N\sqrt{N})$	✓	FP
Compressive Transformer† [59]	$O(N^2)$	✓	RC
Sinkhorn Transformer† [74]	$O(B^2)$	✓	LP
Longformer [5]	$O(n(k+m))$	✓	FP+M
ETC [3]	$O(N_g^2 + NN_g)$	×	FP+M
Synthesizer [73]	$O(N^2)$	✓	LR+LP
Performer [10]	$O(N)$	✓	KR
Funnel Transformer [15]	$O(N^2)$	✓	FP+DS
Linformer [87]	$O(N)$	×	LR
Linear Transformers† [36]	$O(N)$	✓	KR
Big Bird [93]	$O(N)$	×	FP+M
Random Feature Attention† [56]	$O(N)$	✓	KR
Long Short Transformers† [96]	$O(kN)$	✓	FP + LR
Poolingformer† [95]	$O(N)$	×	FP+M
Nyströmformer† [91]	$O(kN)$	×	M+DS
Perceiver [31]	$O(kN)$	✓	M+DS
Clusterformer† [88]	$O(N \log N)$	×	LP
Luna [50]	$O(kN)$	✓	M
TokenLearner† [63]	$O(k^2)$	×	DS
Adaptive Sparse Transformer† [14]	$O(N^2)$	✓	Sparse
Product Key Memory [41]	$O(N^2)$	✓	Sparse
Switch Transformer [23]	$O(N^2)$	✓	Sparse
GShard [45]	$O(N^2)$	✓	Sparse
Scaling Transformers [32]	$O(N^2)$	✓	Sparse
GLaM [21]	$O(N^2)$	✓	Sparse

Models in the first section are mainly efficient attention methods. Models in the subsequent lower section generally refer to sparse models. Articles annotated with a superscript † are peer-reviewed articles. Class abbreviations include: FP = Fixed Patterns or Combinations of Fixed Patterns, M = Memory, LP = Learnable Pattern, LR = Low-Rank, KR = Kernel RC = Recurrence, and DS = Downsampling. Furthermore, N generally refers to the sequence length and B is the local window (or block) size. N_g and N_c denote global model memory length and convolutionally compressed sequence lengths, respectively.

- *Combination of Patterns (CP).* The key idea of combined[2] approaches is to improve coverage by combining two or more distinct access patterns. For example, the Sparse Transformer [9] combines strided and local attention by assigning half of its heads to each pattern. Similarly, Axial Transformer [28] applies a sequence of self-attention computations given a

[2]We note that this is also often referred to as *factorization approaches*, e.g., in Child et al. [9]. 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.

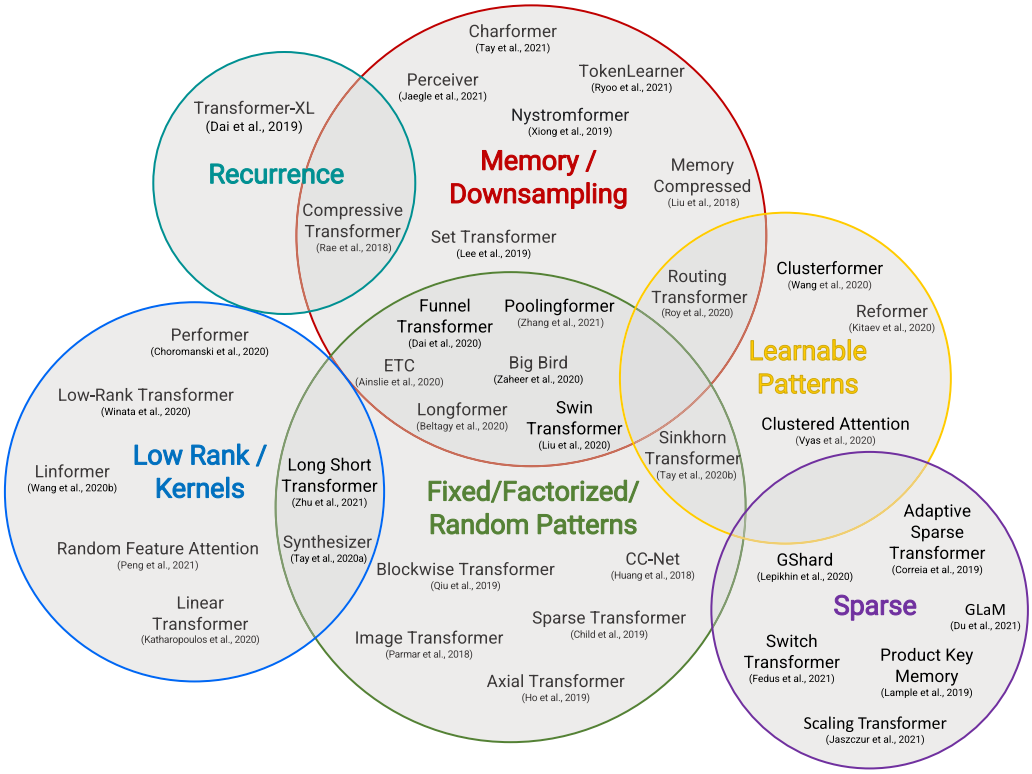


Fig. 2. Taxonomy of Efficient Transformer Architectures.

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 [84, 88]. Notably, Reformer [37] introduces a hash-based similarity measure to efficiently cluster tokens into chunks. In a simlar vein, the Routing Transformer [62] employs online k-means clustering on the tokens. Meanwhile, the Sinkhorn Sorting Network [74] 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 learns to sort/cluster the input tokens—enabling a more optimal global view of the sequence while maintaining the efficiency benefits of fixed patterns approaches.
- *Neural Memory.* Another prominent method is to leverage a learnable side memory module that can access multiple tokens at once. A common form is *global* neural[3] memory which is able to access the entire sequence. The global tokens act as a form of model memory that

[3]We use the term *neural* here to refer to a representation-like memory that is often manifested in the model.

learns to gather from input sequence tokens. This was first introduced in Set Transformers [43] 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 [71]. Global memory tokens are also used in ETC [3] and Longformer [5]. With a limited amount of neural 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 [87] is a 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 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 [10, 36] 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 type of low-rank approach [10]. Examples of recent work in this area include Performers, Linear Transformers, and Random Feature Attention (RFA [56]).
- *Recurrence.* A natural extension to the blockwise method is to connect these blocks via recurrence. Transformer-XL [16] 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.
- *Downsampling.* Another popular method of reducing computation cost is to reduce the resolution of the sequence, hence reducing computation costs by a commensurate factor. Examples of this class of models include Perceiver [31], Funnel Transformers [15], Swin Transformer [49], and Charformer [78] models. Notably, there might also be some form of overlap of this class of models with models that leverage *memory* tokens as models such as Set Transformer can also be viewed as a form of downsampling, albeit within the attention mechanism. The recent Nyströmformer [91], on the surface, may seem like a low-rank or kernal-based approach. However, it is actually a downsampling approach where the *'landmarks'* are simply strided based pooling—in similar spirit to Set Transformer, Funnel Transformer, or Perceiever.
- *Sparse Models and Conditional Computation.* While not targeted specifically at the attention modules, sparse models *sparsely* activate a subset of the parameters which generally improves the parameter to FLOPs ratio. Examples of this class of model includes Switch Transformers [23], GShard [45], Product-Key Memory Layers [41]. Within the scope of our studied models, sparse models typically operate on an adaptive basis in which the sparsity is typically learned (via mixture-of-experts like mechanism). Within this context, we can also consider sparsification of attention weights to fall under this paradigm. For this reason, we believe there is a close connection to fixed or learned patterns in attention. However, we believe that the emergence of an entire research direction [21, 45, 46, 61] based on sparse efficient should warrant a new category of efficient Transformers.

We note that these buckets are a broad characterization of the different efficient Transformer models. In reality, there is no sharp boundary between the buckets as models may be comprised of

multiple technical innovations. For example, the k-means clustering in Routing Transformer [62] can also be interpreted as a form of global model memory approach, since one can view the centroids as parameterized model memory. In Reformer, however, clustering is used to learn the sparsity pattern of the attention weights. Additionally, pooling [48] can be also interpreted as a form of model memory mechanism. We also note that the recent xformer models (circa December 2021) have started adopting some form of two-staged attention mechanism. Many times, these attention mechanisms explicitly combine one or more flavors of the above, e.g., local windows and then memory in Poolingformer [95], or Long Short Transformers [96] that utilize low rank attention with fixed windows (e.g., a combination of local attention with Linformer-like inductive bias).

3.2 Detailed Walk-Through of Efficient Transformer Models

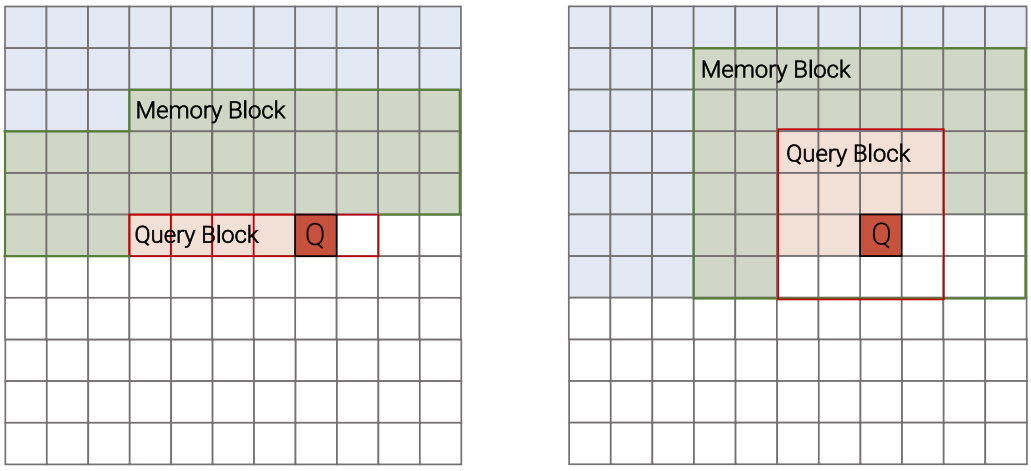
This section delves into the details of several key efficient Transformer models, discussing their pros, cons, and unique talking points. The goal here is not to exhaustively detail all such models, but rather to cover a representative sample of models.

Structure of This Section. We begin by discussing local and fixed patterns models such as the Memory Compressed Transformer [48] and Image Transformer [55]. We then discuss the Set Transformers [43], an early approach for utilizing global model memory. Following which, we move on to models that utilize combinations of patterns such as Sparse Transformers [9], CCNet [30], and Axial Transformers [28]. Next, we discuss Longformer [5] and ETC [3], as examples of memory-based Sparse Transformer approaches. Our detailed walkthrough then moves on to models that incorporate learnable patterns (LP) such as Routing Transformers [62], Reformer [37] and Sinkhorn Transformers [74]. After which, we introduce Linformer [87] and Synthesizers [73], models that can be considered low-rank factorization approaches. We then discuss models based on kernel approaches such as Performer [10] and Linear Transformers [36]. Following which, we discuss the models that are based on segment-based recurrence such as Transformer-XL [16] and Compressive Transformers [59]. Finally, we discuss the family of Sparse models which primarily leverage Mixture-of-Experts (MoE) type architectures and conditional computation to achieve computational efficiency. The logical flow of this section is aimed to be loosely chronological instead of categorically organized (with the exception of certain buckets like recurrence or sparsity that are more orthogonal approaches). We believe this is pedagogically helpful.

3.2.1 Memory Compressed Transformer. Memory Compressed Transformer [48] is one of the early attempts at modifying Transformers to better handle longer sequences. The modification introduced by Memory Compressed Transformers is in two folds: localizing the attention span and using memory compressed attention.

Local Attention Span. A straightforward solution for dealing with long sequences in Transformers is to limit the attention span to a local neighborhood. Liu et al. [48] proposed dividing the input sequence into blocks of similar length so that self-attention can be computed within each block independently. This keeps the cost of attention per block constant, thus the number of activations scales linearly with the input length.

Memory-Compressed Attention. The idea behind memory compressed attention is to reduce the number of keys and values using a strided convolution, while the queries remain unchanged. This leads to a reduction in 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 local attention.



(a) 1-dimensional local attention (b) 2-dimensional local attention

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

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

3.2.2 Image Transformer. Image Transformer [55], 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 locality 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 [54, 55] addresses the issues with the computational and memory costs 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[4] and partitioned into non-overlapping query blocks Q of length l_q, and for each query block, a memory block M is built from the same pixels in the Q as well as a fixed number of pixels, l_m, generated before the query pixel. In 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

[4]Given a 2D image as a grid of pixels, the horizontally left-to-right scanning of pixels, line-by-line, creates a raster order.

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. Figure 1 illustrates an overview of the Transformer architecture.

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 $O(n \cdot m)$.

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 has quadratic complexity with respect to the region length, thereby introducing an extra hyper-parameter in the trade-off between performance and computational complexity.

3.2.3 Set Transformer. The Set Transformer [43] 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. [94] 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* ρ (pool$(\cdot)$). While this form is a universal approximator in the space of permutation-invariant functions, it is unclear how well such models fit tasks in practice. The Set Transformer proposes a solution that can be viewed as an encoder and 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

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

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:

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

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 model memory. In addition to reducing the $O(Nn^2)$ complexity of the self-attending SAB layer to $O(Nmn)$, 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 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 model memory in a similar light, reducing the memory and runtime complexity of the architecture.

3.2.4 Sparse Transformer. The Sparse Transformer [9] 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 neighborhoods. 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.

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

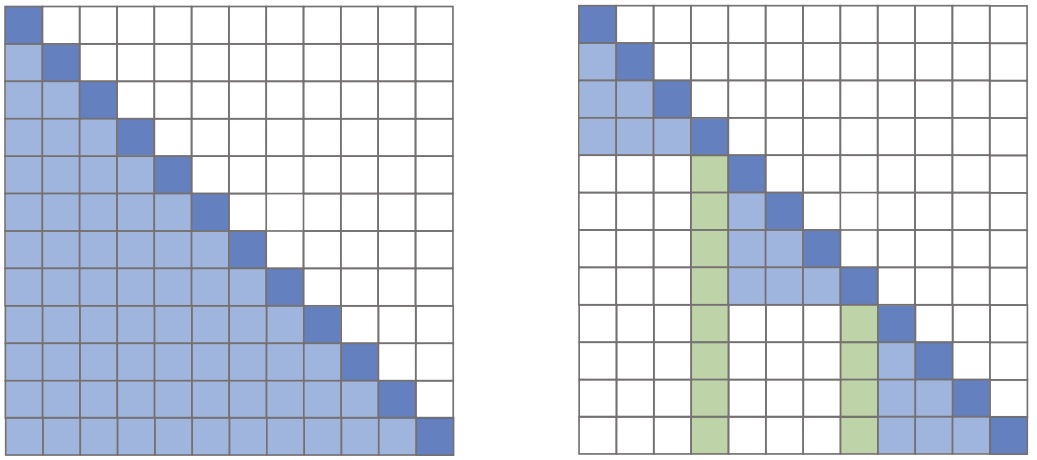
where A_{ij} is the attention weight of q_i, k_j and $\lfloor\ \rfloor$ denote the floor operation. In this case, we only compute the attention if $\lfloor j/N \rfloor = \lfloor i/N \rfloor$ (within the same block).

Strided Attention Heads. The other half of the heads are dedicated to fixed strided patterns. Concretely,

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

The final result of the factorized sparse attention is visualized in Figure 4. We refer interested readers to [92] for some additional theoretical analysis about the expressiveness of the Sparse attention mechanism.

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 $O(n^2)$ to $O(n \log n)$.



(a) Transformer (b) Sparse Transformer

Fig. 4. Illustration of patterns of the attention matrix for dense self-attention in Transformers and sparse fixed attention in Sparse Transformers. Blue in the right diagram represents the local self-attention while green represents the strided component of the sparse attention.

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 [28, 89] 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. Figure 3 provides an illustration of attention applied to 2D input in Image Transformer.

Axial Transformer offers an encoder-decoder architecture. For the decoding, to be able to implement the causal 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 and does not require a custom kernel for an efficient implementation.

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 $O(n\sqrt{n})$, which saves $O(\sqrt{n})$ 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

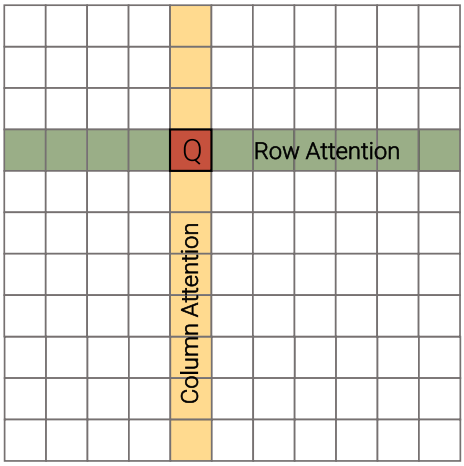


Fig. 5. Attention span in Axial Transformer on a two-dimensional input.

complexity $O(b.b^2)$. In a more general case, for a d-dimensional tensor of shape $N = N^{1/d} \times \cdots \times N^{1/d}$, Axial Transformer saves a $O(N^{(d-1)/d})$ factor of resources over standard self-attention.

3.2.6 Longformer. Longformer [5] 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, Longformer adopts global memory tokens that have access to all input sequences.

Parameter and Memory Complexity. The complexity of the model is reduced from $O(n^2)$ to $O(nk)$ 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 [3] 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 Longformer in the way it introduces global auxiliary tokens. These tokens are trainable parameters and can be interpreted as a form of model memory that pools across the sequence to collect global sequence information.

Memory and Parameter Complexity. The memory complexity of the ETC model is $O(n_g^2 + n_g N)$, 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 Big Bird. The Big Bird model [93] is another Transformer for modeling longer sequences and is primarily built on top of ETC [3]. The Big Bird model is comprised 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 model memory can be traced all the way back to Longformer/ETC and Set Transformer model. Notably, the global model memory in Big Bird is extended to contain tokens within the sequence, instead of simply parameterized model 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 model-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 Big Bird, 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 Big Bird model does not introduce new parameters beyond the Transformer model.

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

3.2.9 Routing Transformer. The Routing Transformer [62] 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 each 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 [71].

Routing Strategy. The routing strategy is then 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 $k \times d$ centroid vectors and a W_r projection matrix. The memory complexity is $O(n^{1.5})$.

3.2.10 Reformer. Reformer [37] is another efficient attention model based on **locality sensitive hashing (LSH)**. Reformer also introduces *reversible* Transformer layers, which contribute 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, Reformer assigns and maintains a position index for every query and 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 the 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 Reformer is $O(n \log n)$. In terms of parameter costs, 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, 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 [74]. 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 [1, 67] 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 in Adams and Zemel [1], and Tay et al. [74].

Parameter and Memory Complexity. The memory complexity of the Sinkhorn Transformer is $O(b^2)$ 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. Linformer [87] is an efficient Transformer based on the idea of low-rank self-attention.

Low-Rank Projections on Length Dimensions. 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. This can 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 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 [35]. 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. The formulation of Linformer (for each attention head) can be expressed as:

$$Softmax\left(\frac{1}{\sqrt{d_k}} XW_i^Q \left(E_i XW_i^K\right)\right) \cdot F_i XW_i^V \tag{5}$$

where $W^{Q,K,V}$ are the default linear transformation of X into queries (as per vanilla Transformer) and E_i, F_i are additional $k \times N$ projection of the key and values into $k \times d$ tensors.

Parameter and Memory Complexity. The memory complexity of Linformer is $O(n)$. 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 Performer. The Performer [10, 11] 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:

$$A = [g(Q_i^\top)K(Q_i^\top K_j^\top)h(K_j^\top)] \tag{6}$$

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:

$$Y = \hat{D}^{-1}(Q'((K')^\top V)) \tag{7}$$

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:

$$\phi(X) = \frac{c}{\sqrt{M}} f(Wx + b)^\top, \tag{8}$$

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 [10].

Parameter/Memory Complexity and Compute Costs. The complexity of the bi-directional FAVOR algorithm is $O(Md + Nd + MN)$ where M is the dimensionality of the random features. It is worth noting that the unidirectional variations cannot be causally masked in an efficient linear-time fashion. As such, during training, running unidirectional (causal) implementation of kernel-based attention on an autoregressive task can be several times slower than vanilla Transformer during *parallelized* training due to the need to do a left to right pass (i.e., scan operation) in similar spirit to Recurrent neural networks. Since many autoregressive tasks trained via parallelization and teacher forcing, this makes training Performer on a generative task prohibitively slow. In order for KV to be causally masked efficiently, one would have to manifest the $d \times d$ KV matrix at every time step—recovering a quadratic complexity model. We feel this is one of the intricate points that highlight how efficient memory complexity might not equate a faster or more efficient model in practice. We highlight that this only happens during autoregressive training. The inference-time for incremental decoding, however, would benefit from a speed-up.

3.2.14 Linear Transformer. The Linear Transformer [36] 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 causal 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. Linear Transformers are similar to Performers with the exception of the kernel function and therefore also suffer from the same drawbacks (unable to be parallelized across the time dimension during training in an autoregressive teacher forced setting).

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, $\text{sim}(q, k) = \exp(\frac{q^T k}{\sqrt{d}})$. 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:

$$V_i' = \frac{\phi(Q_i)^T S_p}{\phi(Q_i)^T Z_p},$$

$$S_p := \sum_{j=1}^{p} \phi(K_j)V_j^T,$$

$$Z_p := \sum_{j=1}^{p} \phi(K_j).$$

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:

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

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

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

where elu$(\cdot)$ denotes the exponential linear unit [13]. With this choice of feature map, $c = d$ and the end-to-end complexity of the model is $O(Nd^2)$.

3.2.15 Synthesizers. Synthesizer models [73] are an attempt to study and investigate the true importance of conditioning within the self-attention mechanism and are also the first attempts at unconditional token-mixing. In Tay et al. [73], 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 and can be considered more as a MLP-Mixer [81]. 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:

$$A = W_2(\sigma_R(W_1(X) + b)) + b, \tag{9}$$

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:

$$Y = \text{Softmax}(A)G(X), \tag{10}$$

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

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

$$Y = \text{Softmax}(R)G(X), \tag{11}$$

where $R \in \mathbb{R}^{N\times N}$ is a trainable and/or non-trainable matrix. In Tay et al. [73], 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, the output can be written as:

$$Y = \text{Softmax}(R_1R_2^\top)G(X), \tag{12}$$

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

$$A = H_B(B) * H_C(C) \text{ where } B, C = F_B(X_i), F_C(X_i), \tag{13}$$

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 [16] 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:

$$\tilde{\boldsymbol{h}}_{\tau+1}^{n-1} = \left[\text{SG}\left(\boldsymbol{h}_{\tau}^{n-1}\right) \odot \boldsymbol{h}_{\tau+1}^{n-1}\right] \tag{14}$$

$$q_{\tau+1}^{n}, k_{\tau+1}^{n}, v_{\tau+1}^{n} = \boldsymbol{h}_{\tau+1}^{n-1}\boldsymbol{W}_{q}^{\top}\ ,\ \tilde{\boldsymbol{h}}_{\tau+1}^{n-1}\boldsymbol{W}_{k}^{\top}\ ,\ \tilde{\boldsymbol{h}}_{\tau+1}^{n-1}\boldsymbol{W}_{v}^{\top} \tag{15}$$

$$\boldsymbol{h}_{\tau+1}^{n} = \text{Transformer}\left(q_{\tau+1}^{n}, k_{\tau+1}^{n}, v_{\tau+1}^{n}\right), \tag{16}$$

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. [16] for more details.

3.2.17 Compressive Transformers. Compressive Transformers [59] 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.

Model Memory. The Compressive Transformer is characterized by a dual model memory system—a primary model memory and a secondary compressed model memory. It maintains a model 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 model memory are moved to the compressed model 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 model memory instead of the lossless reconstruction of the model memory itself.

3.2.18 Sparse Models. In this section, we describe the family of Sparse models. Sparse models typically achieve a high parameter to FLOP ratio by sparsely activating a subset of parameters

or activations. It is good to note that while most of the works within the scope of this survey deals with efficient attention, the scope of sparse models goes beyond the attention module and is generally applied more frequently to the feed forward layers [23, 45]. In this section, we discuss the prime variant for Sparse models, i.e., the Mixture-of-Experts–based Sparse models which includes models such as GShard [45], Switch Transformer [23], and GLaM [21].

Mixture-of-Experts. The key idea behind MoE is to route token x_i to a set of selected experts determined by a routing function. The routing function typically computed a linear combination over experts using the softmax function and can be interpreted as a form of gating mechanism. The top-k gate values are then selected for each token x_i and the final output of that layer is determined by a linear combination of selected top-k experts. This MoE layer remains foundational and fundamental to many MoE architectures, with the exception of certain implementation details. For example, Switch uses a top-1 routing strategy while GShard uses a group-level top-2 gating.

4 DISCUSSION

This section explores 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 not 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 [20] 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 [9], Adaptive Transformers [14], Routing Transformers [62], and Reformers [37] are mainly focused on generative modeling tasks. These benchmarks typically involve language modeling and/or pixel-wise image generation on datasets such as wikitext [51], and/or ImageNet [19]/ CIFAR [38]. 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 the 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 [3] only runs experiments on question-answering benchmarks such as NaturalQuestions [39] or TriviaQA [34]. On the other hand, the Linformer [87] focuses on subsets of the GLUE [85] benchmark. This split is very natural and intuitive, since models like ETC and Linformer cannot be used in an auto-regressive fashion. This exacerbates the challenges associated with comparing these encoder-only models with the other models.

There are models that focus on a balance of both. Longformer [5] tries to balance this by running benchmarks on both generative modeling and encoder-only tasks. The Sinkhorn Transformer [74] compares on both generative modeling tasks as well as encoding only tasks.

Additionally, it is also worth noting that, although Seq2Seq **machine translation (MT)** was one of the problems that popularized Transformer models, not many of these efficient Transformer models are evaluated on MT tasks. This is likely because sequence lengths in MT are 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 articles choose to evaluate on. For starters, the Performer model [10] evaluates on masked language modeling on proteins, deviating from serious head-on comparisons with other efficient Transformer models. The Linear Transformer [36] also evaluates on speech recognition, which is a rare benchmark amongst this group of papers.

There have been recent attempts to unify evaluation on Efficient Transformers, namely **Long-Range Arena (LRA)**, [75] that benchmarked 10 different xformer variants on long-range modeling tasks. It is good to note that LRA was designed for evaluating Transformers in encoder-only mode and do not consider generative (or autoregressive tasks) that require causal masking.

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 primarily been focused on more intuitive and simple approaches such as *fixed patterns*. To this end, most early work in this area is based on block/local patterns such as Image Transformer [55], Compressed Attention [48], Blockwise Transformer [58] or the local windows in Sparse Transformer [9].

The paradigm of factorizing various fixed patterns was first introduced in Child et al. [9] and CCNet [30]. Around this same time, we start to observe early traces of *model-memory*-based approaches from both the inducing point methods in the Set Transformer [43] or global nodes in the Star Transformer [25] model.

We observe the next wave of models comes in the form of learnable sparsity patterns. Reformer [37] and Routing Transformers [62] 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 [74] are also based on the idea of sorting, albeit at the 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 [3] and Longformer [5] models are very similar ideas that are fundamentally Sparse Transformer extensions. These models incorporate the notion of a global model memory, which is reminiscent of the Set Transformer's inducing point method or the global model 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 Low-Rank Transformer [90], Linformer [87], Performer [10] and/or Linear Transformers [36]. 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 model memory based efficient Transformer models.

More recently, there have been more models that propose a two-pronged or two-step attention mechanism combining models from different techniques. The Long Short Transformer [96] is a dynamic form of Linformer combined with Fixed Pattern attention mechanisms. On the other hand, models like Poolingformer also explicitly construct a two-level attention mechanism with techniques reminiscent of memory-based approaches and local attention. Scatter Brain is a new work [8] attempts to unify sparse (fixed pattern) attention with low-rank attention. Two-stage attention mechanisms are also proposed by Luna [50].

On the side, it is important to note that the recurrent based models (Transformer-XL and Compressive Transformers) seem to operate orthogonally and are not as directly comparable to the other models. We also observe that Sparse models [23, 45] that are not only applicable to attention

modules, are also recently emerging and becoming more popular and have demonstrated considerable success in the recent months [21].

4.3 Brief Discussion on Orthogonal Efficiency Efforts

While this article is mainly focused on (1) the computational and memory complexity of the self-attention module and (2) sparsity and adaptive computation, 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 reduce the overall model size. The Universal Transformers [18] tie attention and transition weights across layers. Similarly, Albert [42] does the same parameter sharing across layers. On the other hand, the Quaternion Transformer [79] 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. Q-BERT [66] is a model that quantizes Transformer models to ultra-low precision. Meanwhile mixed precision training [53] is a highly popular technique to reduced the memory costs of training Transformers. Fan et al. [22] applies Quantization Aware training to Transformer models.
- *Inference-Time Efficiency and Network Pruning.* Multiple research directions explore improving the Transformer efficiency at inference time. One prime example is network model. An example is to prune attention heads during inference [52, 83]. This has shown to have minimal degradation of performance on downstream tasks. On the other hand, Lagunas et al. [40] proposes a "block" pruning approach which can make a Transformer 2.4× faster with little loss in predictive performance on language tasks. Another line of work involved fast exit during inference which allows us to exit compute if the model is confident of its predictions [65].
- *Knowledge Distillation.* **Knowledge distillation (KD)** [27] 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 [64], task-specific distillation [72] and TinyBERT [33].
- *Neural Architecture Search (NAS).* Searching for more efficient Transformer architectures is also a common strategy. Guo et al. [26] proposed **Neural Architecture Transformer (NAT)**, using NAS to search for more compact and efficient Transformers by removing redundant operations. Wang et al. [86] 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 [29] 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 [69], MAD-X [57], and HyperGrid [80].
- *Alternative Architectures.* A considerable amount of effort have gone into designing Transformer alternatives. Amongst the many alternatives considered, a prominent line of emerging research belongs to the family of MLP Mixers [81]. Different mixing operations have been proposed, such as the G-MLP [47], FNet [44]. Synthesizers [73], although commonly referred to as an efficient attention method, is also an early manifestation of the mixer line of work, as the random matrices similarly act as an unconditioned mixing operation. A recent promising

line of work, based on Structured State Spaces [24] also demonstrated very promising results on long-range modeling. Last, convolutional models are generally more efficient than Transformers since convolutional kernels operate on a fixed, small local neighborhood around the input token. Tay et al. [76] shows that, when pre-trained, these more efficient convolutional models can sometimes match the predictive performance of Transformer ones.

- *Model-Based Indexing.* An emerging novel paradigm is the notion of neural indexing. Differentiable Search Index [77] proposed to index documents within model parameters. This is essentially linking textual content into a *memory address* within model parameters which could be invoked later by referencing this address token.

5 CONCLUSION

In this article, we surveyed the literature on efficient Transformer models especially pertaining to 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 characterized the existing models based on techniques and provided a comprehensive walkthrough of 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

[1] Ryan Prescott Adams and Richard S. Zemel. 2011. Ranking via sinkhorn propagation. *arXiv preprint arXiv:1106.1925* (2011).

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

[3] Joshua Ainslie, Santiago Ontanon, Chris Alberti, Philip Pham, Anirudh Ravula, and Sumit Sanghai. 2020. ETC: Encoding long and structured data in transformers. In *Proceedings of EMNLP* (2020).

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

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

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

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

[8] Beidi Chen, Tri Dao, Eric Winsor, Zhao Song, Atri Rudra, and Christopher Ré. 2021. Scatterbrain: Unifying sparse and low-rank attention. In *Thirty-Fifth Conference on Neural Information Processing Systems*.

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

[10] Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Jared Davis, Tamas Sarlos, David Belanger, Lucy Colwell, and Adrian Weller. 2020. Masked language modeling for proteins via linearly scalable long-context transformers. In *Proceedings of ICLR* (2020).

[11] Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. 2020. Rethinking attention with performers. In *Proceedings of ICLR* (2020).

[12] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2022. PaLM: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311* (2022).

[13] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. 2015. Fast and accurate deep network learning by exponential linear units (ELUS). In *Proceedings of ICLR 2016* (2015).

[14] Gonçalo M. Correia, Vlad Niculae, and André F. T. Martins. 2019. Adaptively sparse transformers. In *Proceedings of EMNLP* (2019).

[15] Zihang Dai, Guokun Lai, Yiming Yang, and Quoc V. Le. 2020. Funnel-transformer: Filtering out sequential redundancy for efficient language processing. In *Proceedings of NeurIPS* (2020).
[16] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. 2019. Transformer-XL: Attentive language models beyond a fixed-length context. In *Proceedings of ACL* (2019).
[17] Mostafa Dehghani, Anurag Arnab, Lucas Beyer, Ashish Vaswani, and Yi Tay. 2021. The efficiency misnomer. *arXiv preprint arXiv:2110.12894* (2021).
[18] Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. 2018. Universal transformers. In *Proceedings of ICLR* (2018).
[19] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *Proceedings of the 2009 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 248–255.
[20] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT:: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL* (2018).
[21] Nan Du, Yanping Huang, Andrew M. Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Wei Yu Adams, Orhan Firat, Barret Zoph, Liam Fedus, Maarten Bosma, Zongwei Zhou, Tao Wang, Yu Emma Wang, Kellie Webster, Marie Pellat, Kevin Robinson, Kathy Meier-Hellstern, Toju Duke, Lucas Dixon, Kun Zhang, Quoc V. Le, Yonghui Wu, Zhifeng Chen, and Claire Cui. 2021. GLaM: Efficient scaling of language models with mixture-of-experts. arXiv:2112.06905 [cs.CL].
[22] Angela Fan, Pierre Stock, Benjamin Graham, Edouard Grave, Remi Gribonval, Herve Jegou, and Armand Joulin. 2020. Training with quantization noise for extreme fixed-point compression. *arXiv preprint arXiv:2004.07320* (2020).
[23] William Fedus, Barret Zoph, and Noam Shazeer. 2021. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *arXiv preprint arXiv:2101.03961* (2021).
[24] Albert Gu, Karan Goel, and Christopher Ré. 2021. Efficiently modeling long sequences with structured state spaces. In *Proceedings of NeurIPS* (2021).
[25] Qipeng Guo, Xipeng Qiu, Pengfei Liu, Yunfan Shao, Xiangyang Xue, and Zheng Zhang. 2019. Star-transformer. In *Proceedings of NAACL* (2019).
[26] Yong Guo, Yin Zheng, Mingkui Tan, Qi Chen, Jian Chen, Peilin Zhao, and Junzhou Huang. 2019. NAT: Neural architecture transformer for accurate and compact architectures. In *Advances in Neural Information Processing Systems*. 737–748.
[27] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* (2015).
[28] Jonathan Ho, Nal Kalchbrenner, Dirk Weissenborn, and Tim Salimans. 2019. Axial attention in multidimensional transformers. *arXiv preprint arXiv:1912.12180* (2019).
[29] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for NLP. In *Proceedings of ICML* (2019).
[30] Zilong Huang, Xinggang Wang, Lichao Huang, Chang Huang, Yunchao Wei, and Wenyu Liu. 2019. CCNET: Criss-cross attention for semantic segmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 603–612.
[31] Andrew Jaegle, Sebastian Borgeaud, Jean-Baptiste Alayrac, Carl Doersch, Catalin Ionescu, David Ding, Skanda Koppula, Daniel Zoran, Andrew Brock, Evan Shelhamer, et al. 2021. Perceiver IO: A general architecture for structured inputs & outputs. *arXiv preprint arXiv:2107.14795* (2021).
[32] Sebastian Jaszczur, Aakanksha Chowdhery, Afroz Mohiuddin, Łukasz Kaiser, Wojciech Gajewski, Henryk Michalewski, and Jonni Kanerva. 2021. Sparse is enough in scaling transformers. *Advances in Neural Information Processing Systems* 34 (2021).
[33] Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2019. TINYBERT: Distilling BERT for natural language understanding. *arXiv preprint arXiv:1909.10351* (2019).
[34] Mandar Joshi, Eunsol Choi, Daniel S. Weld, and Luke Zettlemoyer. 2017. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Vancouver, Canada.
[35] Lukasz Kaiser, Aidan N. Gomez, and Francois Chollet. 2017. Depthwise separable convolutions for neural machine translation. In *Proceedings of ICLR* (2017).
[36] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. 2020. Transformers are RNNs: Fast autoregressive transformers with linear attention. *arXiv preprint arXiv:2006.16236* (2020).
[37] Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. 2020. Reformer: The efficient transformer. In *Proceedings of the International Conference on Learning Representations*. https://openreview.net/forum?id=rkgNKkHtvB.
[38] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. Master thesis.
[39] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Matthew Kelcey, Jacob Devlin, Kenton Lee, Kristina N. Toutanova, Llion Jones, Ming-Wei

Chang, Andrew Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. 2019. Natural questions: A benchmark for question answering research. *Transactions of the Association of Computational Linguistics* (2019).

[40] François Lagunas, Ella Charlaix, Victor Sanh, and Alexander M. Rush. 2021. Block pruning for faster transformers. In *Proceedings of EMNLP 2021* (2021).

[41] Guillaume Lample, Alexandre Sablayrolles, Marc'Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou. 2019. Large memory layers with product keys. *arXiv preprint arXiv:1907.05242* (2019).

[42] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. ALBERT: A lite BERTt for self-supervised learning of language representations. In *Proceedings of ICLR* (2019).

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

[44] James Lee-Thorp, Joshua Ainslie, Ilya Eckstein, and Santiago Ontanon. 2021. FNet: Mixing tokens with fourier transforms. *arXiv preprint arXiv:2105.03824* (2021).

[45] Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. 2020. GSHARD: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668* (2020).

[46] Mike Lewis, Shruti Bhosale, Tim Dettmers, Naman Goyal, and Luke Zettlemoyer. 2021. Base layers: Simplifying training of large, sparse models. *arXiv preprint arXiv:2103.16716* (2021).

[47] Hanxiao Liu, Zihang Dai, David R. So, and Quoc V. Le. 2021. Pay attention to MLPs. In *Proceedings of NeurIPS* (2021).

[48] Peter J. Liu, Mohammad Saleh, Etienne Pot, Ben Goodrich, Ryan Sepassi, Lukasz Kaiser, and Noam Shazeer. 2018. Generating Wikipedia by summarizing long sequences. In *Proceedings of ICLR* (2018).

[49] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. 2021. Swin transformer: Hierarchical vision transformer using shifted windows. *arXiv preprint arXiv:2103.14030* (2021).

[50] Xuezhe Ma, Xiang Kong, Sinong Wang, Chunting Zhou, Jonathan May, Hao Ma, and Luke Zettlemoyer. 2021. LUNA: Linear unified nested attention. In *Proceedings of NeurIPS 2021*.

[51] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2017. Pointer sentinel mixture models. In *Proceedings of ICLR* (2017).

[52] Paul Michel, Omer Levy, and Graham Neubig. 2019. Are sixteen heads really better than one? In *Proceedings of NeurIPS* (2019).

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

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

[55] Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Łukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. 2018. Image transformer. In *Proceedings of ICML 2018* (2018).

[56] Hao Peng, Nikolaos Pappas, Dani Yogatama, Roy Schwartz, Noah A. Smith, and Lingpeng Kong. 2021. Random feature attention. In *Proceedings of ICLR* (2021).

[57] Jonas Pfeiffer, Ivan Vulić, Iryna Gurevych, and Sebastian Ruder. 2020. MAD-X: An adapter-based framework for multi-task cross-lingual transfer. In *Proceedings of EMNLP* (2020).

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

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

[60] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2019. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research, 2020* (2019).

[61] Stephen Roller, Sainbayar Sukhbaatar, Arthur Szlam, and Jason Weston. 2021. Hash layers for large sparse models. *arXiv preprint arXiv:2106.04426* (2021).

[62] Aurko Roy, Mohammad Saffar, Ashish Vaswani, and David Grangier. 2020. Efficient content-based sparse attention with routing transformers. In *Proceedings of TACL* (2020).

[63] Michael S. Ryoo, A. J. Piergiovanni, Anurag Arnab, Mostafa Dehghani, and Anelia Angelova. 2021. TokenLearner: Adaptive space-time tokenization for videos. *Advances in Neural Information Processing Systems (NeurIPS)*.

[64] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. DistilBERT, A distilled version of BERT: Smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108* (2019).

[65] Tal Schuster, Adam Fisch, Tommi Jaakkola, and Regina Barzilay. 2021. Consistent accelerated inference via confident adaptive transformers. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language*

Processing. Association for Computational Linguistics, Online and Punta Cana, Dominican Republic, 4962–4979. https://aclanthology.org/2021.emnlp-main.406.

[66] Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W. Mahoney, and Kurt Keutzer. 2020. Q-BERT: Hessian based ultra low precision quantization of BERT. In *Proceedings of AAAI*.

[67] Richard Sinkhorn. 1964. A relationship between arbitrary positive matrices and doubly stochastic matrices. *The Annals of Mathematical Statistics* 35, 2 (1964), 876–879.

[68] David R. So, Chen Liang, and Quoc V. Le. 2019. The evolved transformer. *Proceedings of ICML* (2019).

[69] Asa Cooper Stickland and Iain Murray. 2019. BERT and pals: Projected attention layers for efficient adaptation in multi-task learning. *Proceedings of ICML* (2019).

[70] Sainbayar Sukhbaatar, Edouard Grave, Piotr Bojanowski, and Armand Joulin. 2019. Adaptive attention span in transformers. *arXiv preprint arXiv:1905.07799* (2019).

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

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

[73] Yi Tay, Dara Bahri, Donald Metzler, Da-Cheng Juan, Zhe Zhao, and Che Zheng. 2020. Synthesizer: Rethinking self-attention in transformer models. In *Proceedings of ICML, 2021* (2020).

[74] Yi Tay, Dara Bahri, Liu Yang, Donald Metzler, and Da-Cheng Juan. 2020. Sparse sinkhorn attention. In *Proceedings of ICML* (2020).

[75] Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. 2021. Long range arena: A benchmark for efficient transformers. *Proceedings of ICLR* (2021).

[76] Yi Tay, Mostafa Dehghani, Jai Gupta, Dara Bahri, Vamsi Aribandi, Zhen Qin, and Donald Metzler. 2021. Are pre-trained convolutions better than pre-trained transformers? *arXiv preprint arXiv:2105.03322* (2021).

[77] Yi Tay, Vinh Q. Tran, Mostafa Dehghani, Jianmo Ni, Dara Bahri, Harsh Mehta, Zhen Qin, Kai Hui, Zhe Zhao, Jai Gupta, et al. 2022. Transformer memory as a differentiable search index. *arXiv preprint arXiv:2202.06991* (2022).

[78] Yi Tay, Vinh Q. Tran, Sebastian Ruder, Jai Gupta, Hyung Won Chung, Dara Bahri, Zhen Qin, Simon Baumgartner, Cong Yu, and Donald Metzler. 2021. Charformer: Fast character transformers via gradient-based subword tokenization. *arXiv preprint arXiv:2106.12672* (2021).

[79] Yi Tay, Aston Zhang, Luu Anh Tuan, Jinfeng Rao, Shuai Zhang, Shuohang Wang, Jie Fu, and Siu Cheung Hui. 2019. Lightweight and efficient neural natural language processing with quaternion networks. In *Proceedings of ACL* (2019).

[80] Yi Tay, Zhe Zhao, Dara Bahri, Donald Metzler, and Da-Cheng Juan. 2020. HyperGrid: Efficient multi-task transformers with grid-wise decomposable hyper projections. In *Proceedings of ICLR* (2020).

[81] Ilya Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Peter Steiner, Daniel Keysers, Jakob Uszkoreit, et al. 2021. Mlp-mixer: An all-mlp architecture for vision. In *Thirty-Fifth Conference on Neural Information Processing Systems*.

[82] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*. 5998–6008.

[83] Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. 2019. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Florence, Italy, 5797–5808. https://doi.org/10.18653/v1/P19-1580

[84] Apoorv Vyas, Angelos Katharopoulos, and François Fleuret. 2020. Fast transformers with clustered attention. In *Proceedings of NeurIPS* (2020).

[85] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*. Association for Computational Linguistics, Brussels, Belgium, 353–355. https://doi.org/10.18653/v1/W18-5446

[86] Hanrui Wang, Zhanghao Wu, Zhijian Liu, Han Cai, Ligeng Zhu, Chuang Gan, and Song Han. 2020. HAT: Hardware-aware transformers for efficient natural language processing. *arXiv preprint arXiv:2005.14187* (2020).

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

[88] Shuohang Wang, Luowei Zhou, Zhe Gan, Yen-Chun Chen, Yuwei Fang, Siqi Sun, Yu Cheng, and Jingjing Liu. 2020. Cluster-former: Clustering-based sparse transformer for long-range dependency encoding. In *Proceedings of ACL-IJCNLP (Findings)* (2020).

[89] Dirk Weissenborn, Oscar Täckström, and Jakob Uszkoreit. 2019. Scaling autoregressive video models. In *Proceedings of ICLR* (2019).

[90] Genta Indra Winata, Samuel Cahyawijaya, Zhaojiang Lin, Zihan Liu, and Pascale Fung. 2020. Lightweight and efficient end-to-end speech recognition using low-rank transformer. In *Proceedings of the 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP-2020)*. IEEE, 6144–6148.

[91] Yunyang Xiong, Zhanpeng Zeng, Rudrasis Chakraborty, Mingxing Tan, Glenn Fung, Yin Li, and Vikas Singh. 2021. Nystr\“ omformer: A Nystr\” om-based algorithm for approximating self-attention. *Proceedings of AAAI* (2021).

[92] Chulhee Yun, Yin-Wen Chang, Srinadh Bhojanapalli, Ankit Singh Rawat, Sashank J. Reddi, and Sanjiv Kumar. 2020. $O(n)$ Connections are expressive enough: Universal approximability of sparse transformers. In *Proceedings of NeurIPS* (2020).

[93] Manzil Zaheer, Guru Guruganesh, Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. 2020. Big bird: Transformers for longer sequences. In *Proceedings of NeurIPS* (2020).

[94] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R. Salakhutdinov, and Alexander J. Smola. 2017. Deep sets. In *Advances in Neural Information Processing Systems*. 3391–3401.

[95] Hang Zhang, Yeyun Gong, Yelong Shen, Weisheng Li, Jiancheng Lv, Nan Duan, and Weizhu Chen. 2021. Poolingformer: Long document modeling with pooling attention. In *Proceedings of ICML* (2021).

[96] Chen Zhu, Wei Ping, Chaowei Xiao, Mohammad Shoeybi, Tom Goldstein, Anima Anandkumar, and Bryan Catanzaro. 2021. Long-short transformer: Efficient transformers for language and vision. *Advances in Neural Information Processing Systems* 34 (2021).

Received 6 April 2021; revised 15 December 2021; accepted 6 April 2022