# A Practical Survey on Faster and Lighter Transformers

QUENTIN FOURNIER, Polytechnique Montréal, Canada
GAÉTAN MARCEAU CARON, Mila - Quebec AI Institute, Canada
DANIEL ALOISE, Polytechnique Montréal, Canada

Recurrent neural networks are effective models to process sequences. However, they are unable to learn long-term dependencies because of their inherent sequential nature. As a solution, Vaswani et al. introduced the Transformer, a model solely based on the attention mechanism that is able to relate any two positions of the input sequence, hence modelling arbitrary long dependencies. The Transformer has improved the state-of-the-art across numerous sequence modelling tasks. However, its effectiveness comes at the expense of a quadratic computational and memory complexity with respect to the sequence length, hindering its adoption. Fortunately, the deep learning community has always been interested in improving the models' efficiency, leading to a plethora of solutions such as parameter sharing, pruning, mixed-precision, and knowledge distillation. Recently, researchers have directly addressed the Transformer's limitation by designing lower-complexity alternatives such as the Longformer, Reformer, Linformer, and Performer. However, due to the wide range of solutions, it has become challenging for researchers and practitioners to determine which methods to apply in practice in order to meet the desired trade-off between capacity, computation, and memory. This survey addresses this issue by investigating popular approaches to make Transformers faster and lighter and by providing a comprehensive explanation of the methods' strengths, limitations, and underlying assumptions.

CCS Concepts: • **Computing methodologies** → **Neural networks**.

Additional Key Words and Phrases: Deep Learning, Efficient Transformer, Self-Attention, Survey

## 1 INTRODUCTION

Sequences arise naturally in a wide range of domains, notably in natural language, biology, and software executions. Rumelhart et al. [97] introduced a family of models called recurrent neural networks (RNNs) based on the idea of parameter sharing to process variable-length sequences. Given an input sequence $\boldsymbol{X}$ comprising $n$ tokens $\boldsymbol{x}^{(i)}$ of dimension $d$, recurrent neural networks iteratively construct a sequence of hidden representations $\boldsymbol{h}^{(i)}$ and produce a sequence of outputs $\boldsymbol{y}^{(i)}$ as illustrated in Figure 1. Unfortunately, vanilla RNNs often suffer from vanishing or exploding gradients, which prevent them from learning long-term dependencies. Hochreiter and Schmidhuber [44] addressed this limitation with the now widely popular long short-term memory (LSTM) network, which circumvents the gradient issues with paths through time. Cho et al. [17] later improved over the LSTM with the simpler gated recurrent unit (GRU).

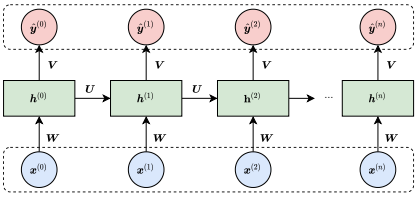


Fig. 1. The computational graph of a recurrent neural network. The input and output sequences are depicted in blue and red, respectively. The position, also known as the time-step, is indicated in superscript. The weight matrices $\boldsymbol{W}$, $\boldsymbol{U}$, and $\boldsymbol{V}$ are shared across all positions. Reproduced with permission [31]. Copyright 2021 IEEE.

Authors' addresses: Quentin Fournier, quentin.fournier@polymtl.ca, Polytechnique Montréal, 2500 Chemin de Polytechnique, Montréal, Quebec, Canada, H3T 1J4; Gaétan Marceau Caron, gaetan.marceau.caron@mila.quebec, Mila - Quebec AI Institute, 6666 Rue Saint-Urbain, Montréal, Quebec, Canada, H2S 3H1; Daniel Aloise, daniel.aloise@polymtl.ca, Polytechnique Montréal, 2500 Chemin de Polytechnique, Montréal, Quebec, Canada, H3T 1J4.

Recurrent neural networks align the input and output sequences, that is, there is a one-to-one mapping between the two sequences. Depending on the task, this property of RNNs may be too restrictive: for instance, translation requires outputting a sequence whose size is often different from that of the input while aligning tokens at different positions. Sutskever et al. [112] addressed this limitation by introducing the sequence-to-sequence framework in which a first network (encoder) processes the entire input sequence and returns its last hidden representation $\boldsymbol{h}^{(n)}$, effectively encoding the input into a fixed-size vector called context. The context then serves as the initial state for a second network (decoder), which generates the output sequence in an autoregressive manner. The decoding stops when a special end-of-sequence token is generated. Figure 2 illustrates the sequence-to-sequence framework.

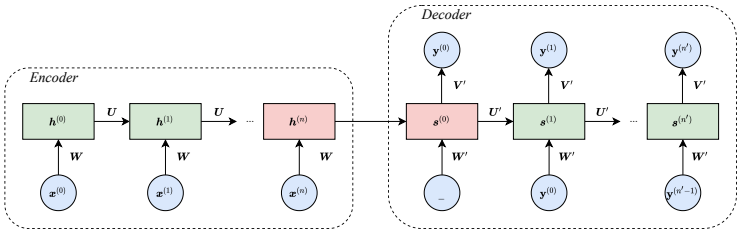


Fig. 2. The sequence-to-sequence framework where the encoder and decoder are recurrent neural networks. The input sequence (blue) is encoded into a fixed-size context $\boldsymbol{h}^{(n)}$ (red), which serves as the initial state of the decoder. Reproduced with permission [31]. Copyright 2021 IEEE.

In practice, the fixed-size nature of the hidden representation hinders the effectiveness of recurrent neural networks [15]. Indeed, as the input sequence is processed, information is iteratively stored into the hidden representation that may be too small to retain all the relevant information for the task. In that case, useful data is inevitably lost, which may significantly impact the model's performance. Bahdanau et al. [3] introduced an alignment mechanism called inter-attention to overcome the bottleneck of the sequence-to-sequence framework. This attention mechanism computes a different representation of the input for each output step, effectively allowing the decoder to "look at" the relevant part(s) of the input for each output step. Thereby, the inter-attention alleviates the encoder's burden to encode all information about the input sequence into a fixed-size vector. Formally, the context is the weighted sum of the encoder's hidden representations $\boldsymbol{h}_i$, for $i = 1, \ldots, n$, where the weights are computed with a feed-forward neural network. For a comprehensive survey of the attention mechanism, we refer the reader to Galassi et al. [33] and Weng [130]. Figure 3 illustrates the inter-attention mechanism.

Moreover, recurrent neural networks do not scale efficiently to longer sequences due to their iterative nature [121]. In particular, RNNs struggle to learn dependencies between distant positions. One measure of this limitation is the relative effective context length (RECL) introduced by Dai et al. [22]. The RECL is the largest context length that leads to a substantial relative gain over the best model. In other words, increasing the context length over the RECL yields a negligible increase in performance over the best model. The authors estimated that the relative effective context length of LSTMs on natural language data is limited to approximately 400 words. Besides, Khandelwal et al. [58] empirically observed that LSTMs sharply model recent positions but only vaguely remember the distant past.

### 1.1 Transformer

This inherent limitation of recurrent neural networks has prevented them from being successfully applied to domains that require processing long sequences such as DNA. To overcome this limitation, Vaswani et al. [121] introduced the *Transformer*, a sequence-to-sequence model built without recurrences. Instead, the Transformer relies solely on the attention mechanism: the inter-attention between the encoder and decoder (see Figure 3), and the self-attention, also known as intra-attention, within the encoder and decoder. The self-attention's main advantage is its ability to relate any two positions of the input sequence regardless of their distance, thus increasing performance significantly on a wide range of tasks, including natural language processing (NLP) [10, 24, 121], computer vision [12, 27, 57], speech recognition [40, 110, 140], and biological sequence analysis [139]. Karita et al. [55] evaluated a Transformer against a sequence-to-sequence Bi-LSTM baseline on automatic speech recognition (ASR), speech translation (ST), and text-to-speech (TTS). The attention-based models outperformed the baseline on 13 corpora out of 15 for monolingual ASR and realized more than 10% relative improvement in 8 languages out of 10 for multilingual ASR. The Transformer improved the BLEU score from 16.5 for the baseline to 17.2 on ST while performing on par for TTS. Table 1 reports the performance improvements brought by popular Transformer architectures over previous state-of-the-art models across different domains. As of this paper's writing, the Transformer has become the de facto model for numerous sequence processing tasks.

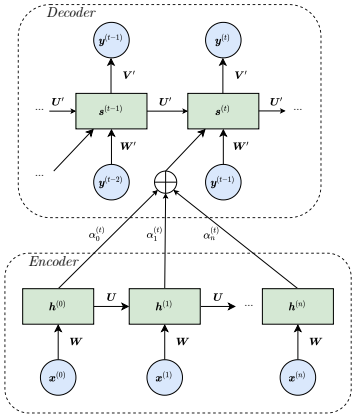


Fig. 3. The inter-attention mechanism. The attention weight $\alpha_i^{(t)}$ depicts the strength with which the $i$-th encoder hidden representation $h^{(i)}$ contributes to the context of $t$-th decoder step. Reproduced with permission [31]. Copyright 2021 IEEE.

As an illustration of an end-to-end application of the Transformer, let us consider the speech recognition task. In hybrid approaches, the recognition system consists of independently trained machine learning components, often an acoustic model, a pronunciation model, and a language model. Instead, in end-to-end approaches, the recognition system consists of a single model comprising several parts trained together. Zhang et al. [140] introduced an end-to-end speech recognition model based on Transformer encoders called the Transformer Transducer that outperformed previous hybrid and end-to-end approaches on the LibriSpeech benchmarks.

The Transformer's capacity comes at the cost of a quadratic computational and memory complexity with respect to the sequence length. Therefore, training large Transformers is prohibitively slow and expensive. For instance, Liu et al. [74] introduced RoBERTa, which was pre-trained on

Table 1. Relative improvements brought by popular Transformer architectures over previous state-of-the-art models. Absolute differences are reported between parenthesis. Sources are: [121] for machine translation, [27] for image classification, [24, 91] for text classification, and [69] for speech-to-text.

| Task | Dataset | Previous SOTA | Transformer's Architecture | Relative Improvement |
|---|---|---|---|---|
| Machine Translation | newstest2014 (EN-to-DE) | MoE (GNMT) [103] | Vanilla [121] | 9.1% (+2.37 BLEU[3]) |
| | newstest2014 (EN-to-FR) | | | 3.1% (+1.24 BLEU) |
| Image Classification | ImageNet | Noisy Student (EfficientNet-L2) [134] | ViT [27] | 0.2% (+0.15% Acc) |
| | CIFAR-10 | BiT-L (ResNet152x4) [60] | | 0.1% (+0.13% Acc) |
| | CIFAR-100 | | | 1.1% (+1.04% Acc) |
| | VTAB (19 tasks) | | | 1.8% (+1.34% Acc) |
| Text Classification | SST2 | Sparse byte mLSTM [39] | BERT[24] | 1.8% (+1.70% Acc) |
| | CoLA | Single-task BiLSTM + ELMo + Attn [124] | | 72.9% (+25.5 MC[4]) |
| Speech-to-text | librispeech (test-clean) | LAS (LSTM) [13, 86] | Convformer [40] | 13.6% (-0.3 WER[5]) |
| | librispeech (test-other) | | | 25.0% (-1.3 WER) |

1024 high-end V100 graphics processing units (GPUs) for approximately a day. Although numerous large pre-trained Transformers have been publicly released, fine-tuning them on the tasks of interest is still computationally expensive. Furthermore, the sequence lengths are restricted by the amount of memory available. Indeed, practitioners typically use large mini-batches with relatively short sequences because the Transformer's optimization is known to be particularly unstable with small mini-batches. Typically, a GPU with 16 GB of memory handles sequences up to 512 words. Consequently, there exists an actual need for lighter and faster Transformers as only a few large organizations can afford to train massive models. As of the writing of this paper, the largest dense Transformer is GPT-3 [10] which requires 355 years to train on a V100 GPU, costing around 4,600,000$ of cloud instances[1].

### 1.2 Lighter and Faster Transformers

Over the years, numerous approaches have been proposed to reduce the computational and memory costs of neural networks, many of which have been applied to Transformers. In this paper, such methods are referred to as *general* since they apply, and have been applied, to a wide range of models. General methods are often orthogonal, and consequently, several of them may be combined to precisely fine-tune the network's capacity, computational cost, and memory usage. However, general methods may be insufficient as the model complexity typically remains unchanged. Therefore, many works introduced lower-complexity variations of the Transformer, referred to as x-formers. In this survey, the Transformer's alternatives are categorized depending on whether they sparsify the attention, factorize it, or modify the network's architecture. Please note that this survey aims to provide a comprehensive summary of the methods that improve the Transformer's efficiency and that fine-grained taxonomies have already been proposed by Tay et al. [116] and Lin et al. [68]. Accordingly, our taxonomy will remain purposefully coarse.

Recently, Tolstikhin et al. [118] and Liu et al. [70] amongst others argued that the powerful yet expensive self-attention mechanism is not necessary to achieve state-of-the-art results and thus challenged the preconception that the self-attention is the source of the Transformer's success. Consequently, they introduced networks without self-attention that are competitive with Transformers for image classification and language modelling at the same computational cost. Yu et al. [137] expanded on this idea with a more general and flexible architecture called MetaFormer where the mechanism to relate the tokens is not specified while the other components are kept the same

[1] https://lambdalabs.com/blog/demystifying-gpt-3
[2] Bilingual evaluation understudy (BLEU), higher is better.
[3] Matthews correlation (MC) coefficient, higher is better.
[4] Word error rate (WER), lower is better.

as the Transformer. Despite the recent success of attention-free architectures, such networks are outside the scope of this paper as they arguably remove the Transformer's core mechanism and are discussed in appendix.

The remainder of this survey is organized as follows. Section 2 introduces the Transformer's architecture and the origin of the quadratic complexity. Section 3 investigates the popular general methods that have been applied to Transformers to reduce the computations and memory footprint. Section 4 explores the recent lower-complexity Transformers. Section 5 explains the limitations of the different approaches and the current evaluation methodology, Section 6 provides a discussion on the broader impact of lighter and faster Transformers, and Section 7 points out potential future research directions. Finally, Section 8 concludes this survey. Practitioners and researchers can find detailed practical guidelines regarding the general and specialized methods in appendix, as well as a summary of the specialized methods (see Table 4) and a discussion about some of the most popular attention-free alternatives.

## 2 TRANSFORMER

This section formally introduces the attention mechanism, the Transformer's architecture, and the root cause of its quadratic complexity.

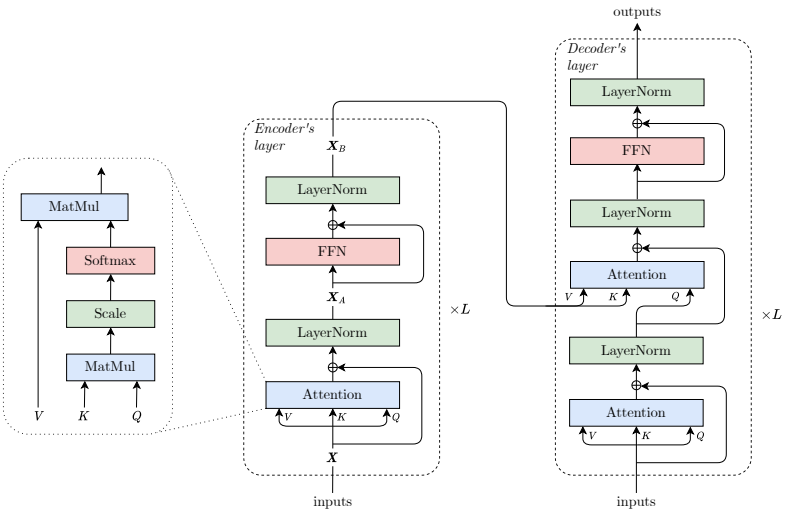


Fig. 4. The Transformer's computational graph [121]. From left to right, the scaled dot product self-attention, the encoder, and the decoder. Note that both the encoder and decoder comprise $L$ identical layers, of which only one is depicted.

### 2.1 Attention Mechanism

The attention mechanism relies on three matrices, namely $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V} \in \mathbb{R}^{n \times d}$, commonly referred to as "queries", "keys", and "values", respectively. The attention outputs the sum of the values weighted by a compatibility or alignment score between each token, which is computed with the function $\text{Score}(\boldsymbol{Q}, \boldsymbol{K}) \in \mathbb{R}^{n \times n}$. Intuitively, if the $i$-th query is highly compatible with the $j$-th key, then the $j$-th value greatly contributes to the $i$-th attention's output. The attention mechanism may be written as:

$$\text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{Score}(\boldsymbol{Q}, \boldsymbol{K})\boldsymbol{V}. \tag{1}$$

Since the compatibility score directly controls the alignment between the tokens, many functions have been proposed. In the original paper, the Transformer relies on the *scaled dot product attention*. The *dot product* refers to the computation of the compatibility score between a single query and a single key. In practice, however, the compatibility scores are computed simultaneously for every query and key by multiplying $\boldsymbol{Q}$ with $\boldsymbol{K}^\top$. Indeed, the $(i, j)$ entry of the $\boldsymbol{Q}\boldsymbol{K}^\top$ multiplication is equal to the dot product between the $i$-th query and the $j$-th key. In order to obtain a probability distribution over the positions, referred to as attention weights, each row of $\boldsymbol{Q}\boldsymbol{K}^\top$ is passed through a Softmax function defined as follows:

$$\text{Softmax}(\boldsymbol{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \quad \text{for } i = 1, \ldots, n. \tag{2}$$

where $\boldsymbol{x} \in \mathbb{R}^n$. Since the dot product grows large in magnitude for large values of $d$, thereby pushing the Softmax into a region of small gradients, a *scaling* factor $\sqrt{d}$ is introduced. Thus, the scaled dot product attention is given by:

$$\text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{Softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d}}\right)\boldsymbol{V}. \tag{3}$$

Nonetheless, the attention presented above may not be flexible enough if the relevant information for the task is scattered across different regions of the input space. That is due in part to the Softmax being exponential, which amplifies the differences between the values. As a result, only a few attention weights are large, i.e., only a few positions are strongly attended. Vaswani et al. [121] addressed this limitation with the multi-head attention. The $d$-dimensional queries, keys and values matrices are first linearly projected $h$ times with distinct, learned projections to $d_k$, $d_k$ and $d_v$ dimensions, respectively. On each projection, an independent attention instance called *head* is applied, and the output of each attention head is concatenated before being linearly projected. The Transformer's multi-head scaled dot product attention is given by:

$$\text{MultiHead}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = [\text{head}_1; \ldots; \text{head}_h]\boldsymbol{W}^O. \tag{4}$$

$$\text{head}_i = \text{Softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{W}_i^Q(\boldsymbol{K}\boldsymbol{W}_i^K)^\top}{\sqrt{d_k}}\right)\boldsymbol{V}\boldsymbol{W}_i^V. \tag{5}$$

where $\boldsymbol{W}_i^Q \in \mathbb{R}^{d \times d_k}$, $\boldsymbol{W}_i^K \in \mathbb{R}^{d \times d_k}$, $\boldsymbol{W}_i^V \in \mathbb{R}^{d \times d_v}$ are the matrices that project the queries, keys, and values into the $i$-th subspace, respectively, and where $\boldsymbol{W}^O \in \mathbb{R}^{hd_v \times d}$ is the matrix that computes a linear transformation of the heads. Typically, $d_k = d/h$ where $d$ is the input and output dimension, and $h$ is the number of heads. For the sake of clarity, methods that modify the attention will be explained in the context of a single head (see Equation 3).

Thus far, the attention mechanism has been described as a general method. The Transformer relies on two specific instances of this mechanism: the intra-attention, popularly known as self-attention, and the inter-attention, sometimes referred to as cross-attention. In the case of inter-attention, the queries correspond to the decoder's hidden representations, and the keys and values are the encoder's outputs. It allows the decoder to look at the relevant parts of the input to produce the output. In the case of self-attention, the three matrices are linear projections of the layer's input, which allows the encoder and decoder to focus on the relevant part of the sequence for each position, similarly to the inter-attention depicted in Figure 3.

## 2.2 Encoder

The Transformer's encoder is a function defined as the composition of $L$ identical layers or blocks, each composed of two sub-layers. The first sub-layer is the aforementioned self-attention mechanism. The second sub-layer is a simple fully connected feed-forward network applied position-wise, that is, independently and identically to every position. The feed-forward network increases the encoder's expressiveness and transforms the self-attention's output for the next layer.

Inspired by ResNet [42], a skip connection, shortcut connection, or residual connection is applied around each sub-layer to create a direct path for the gradient to flow throughout the network. Notably, residual connections make the training of very deep neural networks more stable. Additionally, both sub-layers' outputs are normalized after the residual connection with the layer normalization technique, referred to as LayerNorm [64]. Normalization is a widely adopted technique in deep learning that enables faster and more stable training. Although the rationale behind the normalization's empirical success is not yet fully understood [67], it has been conjectured that this results from a smoother optimization landscape, and to a lesser extent, from a reduction in internal covariance shift [100]. Figure 4 depicts the computational graph of an encoder's layer.

In natural language processing, the input sequence $\boldsymbol{X}$ would typically represent a sentence or a paragraph, and the token $\boldsymbol{x}^{(i)}$ would be its $i$-th word or subword embedding. Each encoder's layer is given by:

$$\boldsymbol{X}_A = \text{LayerNorm}(\text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) + \boldsymbol{X}) \tag{6}$$

$$\boldsymbol{X}_B = \text{LayerNorm}(\text{FFN}(\boldsymbol{X}_A) + \boldsymbol{X}_A) \tag{7}$$

where $\boldsymbol{X}$ and $\boldsymbol{X}_B$ are the layer's input and output, respectively, and $\boldsymbol{Q}$, $\boldsymbol{K}$, and $\boldsymbol{V}$ are linear projections of $\boldsymbol{X}$.

The feed-forward network is given by:

$$\text{FFN}(\boldsymbol{x}) = \max(0, \boldsymbol{x}\boldsymbol{W}_1 + \boldsymbol{b}_1)\boldsymbol{W}_2 + \boldsymbol{b}_2 \tag{8}$$

where $\boldsymbol{W}_1 \in \mathbb{R}^{d \times d_f}$ and $\boldsymbol{W}_2 \in \mathbb{R}^{d_f \times d}$, and where $d_f$ is the dimension of the hidden layer. Note that the feed-forward network is defined for a row vector since it is applied position-wise, that is, it is independently and identically applied to every position or row.

Finally, the position-wise layer normalization is given by:

$$\text{LayerNorm}(\boldsymbol{x}) = \boldsymbol{g} \odot \frac{\boldsymbol{x} - \mu}{\sqrt{\sigma^2 + \epsilon}} + \boldsymbol{b} \tag{9}$$

where $\odot$ denotes the element-wise (Hadamard) product, where the average $\mu$ and the standard deviation $\sigma$ are computed from all of the summed inputs, where the gain $\boldsymbol{g}$ and the bias $\boldsymbol{b}$ are learned parameters of dimension $d$, and where $\epsilon$ is a small constant used in practice for numerical stability.

## 2.3 Decoder

The decoder is also composed of $L$ identical layers. Although it is common for the decoder to have the same number of layers as the encoder, one may adjust their depth independently. Each decoder's layer comprises three sub-layers. The first sub-layer is the self-attention mechanism, as in the encoder, except that future positions are masked. Indeed, while the encoder is allowed to look at future positions since the input sequence is entirely available, the decoder is autoregressive and thus cannot look at future positions since they have not yet been predicted. Therefore, the $i$-th position may only attend to positions less than $i$. The second sub-layer is the inter-attention mechanism, which helps the decoder focus on the relevant parts of the input. Finally, the third

sub-layer is a simple feed-forward network. As for the encoder, a residual connection and a layer normalization are applied to each sub-layer.

Note that the decoder may be safely omitted when the task does not require the sequence-to-sequence framework, such as sentiment analysis, which predicts whether a sentence is positive. One of the most popular encoder-only Transformers is the Bidirectional Encoder Representations from Transformers (BERT) [24], a state-of-the-art language model that learns contextualized embeddings. Nonetheless, autoregressive tasks such as machine translation still require the sequence-to-sequence framework.

### 2.4 Complexity

Intuitively, the quadratic complexity emerges from the computation of the compatibility score between every pair of positions. More precisely, the $\boldsymbol{Q}\boldsymbol{K}^\top$ multiplication requires $n^2$ computations and memory. Such attention is said to be full since any output position is able to attend to any input position. The attention pattern is visualized by means of a connectivity matrix, which indicates the input positions that each output position is able to attend (see Figure 5).

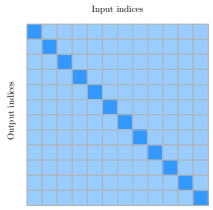


Fig. 5. The connectivity matrix of the full attention. The $i$-th output position attends to the $j$-th input position if, and only if, the cell $(i, j)$ is coloured. The diagonal is highlighted to ease the reading.

What justifies such efforts from the community to improve the Transformer's efficiency? In our opinion, there are three primary motivations: affordability, scalability, and ecology.

The foremost reason is affordability. The Transformer has largely surpassed convolutional and recurrent neural networks and achieved new state-of-the-art results across many tasks. However, those networks have a linear complexity with respect to the sequence length [121], making them affordable to most researchers and practitioners. As explained by Strubell et al. [109], this creates three major issues: (1) it stifles creativity as researchers and practitioners that do not have access to considerable resources are not able to experiment with Transformers, (2) it reinforces the "rich get richer" cycle where successful labs and companies receive more funding due to their existing accomplishments with Transformers, and (3) it forces smaller labs and companies to rely on private cloud services that end up more expensive.

The second reason is scalability. The quadratic complexity prevents researchers and practitioners, even those with access to considerable resources, from applying Transformers on long sequences such as entire chapters or books, high-resolution images or videos, and DNA.

The third reason is ecology. It is now more apparent than ever that we must cut carbon dioxide (CO2) emissions in half over the next decade to limit global warming. The large-scale infrastructures used by the deep learning community consume a considerable amount of electricity, which is mainly produced by non-renewable sources such as coal or gas [49].

Thereby, the following sections investigate popular and novel methods to make Transformers faster and lighter.

## 3 GENERAL APPROACHES

Computational resources have always been a limiting factor for deep learning models [63]. Therefore, numerous approaches have been proposed throughout the years to design faster and lighter models. This section introduces the most popular techniques that apply to virtually all neural networks.

**Gradient Checkpointing** [14]: Intermediate results computed during the forward pass, also referred to as activations, are required to compute the gradients during the backward pass; therefore, they are stored in memory. Activations typically account for most of the memory during training: given an $l$-layer network, the number of intermediate results is proportional to the number of layers ($O(l)$). With gradient checkpointing, also known as rematerialization, activations are stored only for a subset of the layers. However, they must be recomputed during the backward pass, trading memory for computations. In the extreme case where no activations are stored, the memory usage becomes constant ($O(1)$) at the cost of a quadratic number of computations with respect to the number of layers ($O(l^2)$). Chen et al. [14] designed a scheme to select the preserved values that reduces the memory requirement from $O(l)$ to $O(\sqrt{l})$ at the cost of a single additional forward pass per mini-batch. OpenAI implementation of gradient checkpointing [84] obtains an impressive 10× reduction in memory at the cost of a 20% increase in computation time.

**Reversible Layers** [25, 26, 35]: As explained above, the back-propagation requires the activations of all intermediate layers, which are either stored in memory during the forward pass or recomputed during the backward pass. As a solution to the latter case, reversible layers allow their activation to be reconstructed exactly from the next layer; therefore, activations must only be stored for one layer and their memory cost becomes independent of the network's depth. More formally, each reversible layer takes as input $(x_1, x_2)$ and outputs $(y_1, y_2)$ such that $y_1 = x_1 + f(x_2)$ and $y_2 = x_2 + g(y_1)$. Each layer's activations are easily reconstructed as $x_2 = y_2 - g(y_1)$ and $x_1 = y_1 - f(x_2)$.

Kitaev et al. [59] used reversible layers in their Transformer, called the Reformer, by combining the attention and feed-forward sub-layers inside a reversible layer. Specifically, $f(.)$ and $g(.)$ were the Attention(.) and FFN(.) functions, respectively. The authors observed that reversible layers reduced the memory usage of a 3-layer Transformer without degrading its performance. Nonetheless, reversible layers add numerical errors that accumulate over multiple layers and may degrade the model performance. Therefore, they are not suited for very deep networks.

Gradient checkpointing and reversible layers are very much alike in that they trade computations for memory by recomputing activations during backpropagation. This trade-off is sometimes necessary: although computation bottlenecks entail longer running times, memory bottlenecks are critical as they prevent using the model altogether.

**Parameter Sharing**: A simple approach to reduce the number of trainable parameters is to impose sets of parameters to be equal in different parts of the network. In other words, the same parameters are used for multiple operations but need to be stored only once in memory. Such a technique is often referred to as parameter sharing, weight tying, or weight replication. As explained in Section 1 and illustrated in Figure 1, recurrent neural networks are built around this idea of parameter sharing to process variable-length sequences. Parameter sharing has also been applied to Transformers. For instance, the Linformer [126] shares projection matrices across heads and layers, and the Reformer [59] shares its queries and keys parameters, that is, $\boldsymbol{W}^Q = \boldsymbol{W}^K$. Both authors investigated the impact of parameter sharing and concluded that it did not degrade their respective models' performance on their tasks. Lan et al. [62] shared all parameters between layers, which drastically reduced the number of parameters but also decreased the performance by up to 2.5% on average. They observed that sharing only the attention parameters resulted in a slight drop in performance of 0.7% on average. The decrease in performance is to be expected since parameter sharing reduces the number of free parameters, hence the model's capacity.

**Pruning** [63]: Smaller neural networks are not only faster and lighter, but they are also more likely to generalize better than larger models because they presumably extract underlying explanatory factors without redundancy. To reduce the model size, weights with a small saliency, that is, whose deletion have a small effect on the loss, may be removed from large models after training. Methods that consider individual weights are said to be *unstructured*, and methods that consider pieces of the network structure such as attention heads or layers are said to be *structured*. Many structured and unstructured pruning schemes have been proposed, several of which have been applied to Transformers. For instance, Sajjad et al. [98] reduced the size of BERT by 40% by dropping complete layers while retaining between 97 and 98% of its original performance, and Michel et al. [79] pruned away between 20% and 40% of BERT attention heads without any significant loss in performance. Recently, the lottery ticket hypothesis has brought a new justification to pruning neural networks. As introduced by Frankle and Carbin [32], the hypothesis states that a "*randomly-initialized, dense neural network contains a subnetwork that is initialized such that – when trained in isolation – it can match the test accuracy of the original network after training for at most the same number of iterations.*". Prasanna et al. [89] successfully verified this hypothesis on BERT, even noticing that BERT worst subnetworks remain highly trainable. Nonetheless, pruning has two limitations: a large model must be trained, and unstructured pruning schemes produce sparse models unoptimized for modern GPUs and tensor processing units (TPUs).

**Knowledge Distillation** [2, 43]: The knowledge of a large model or an ensemble of models (teacher) is transferred to a single smaller model (student) by training the student to reproduce the teacher's outputs or its internal behaviour. The cumbersome teacher is then discarded, and the student is used at inference time. Given a parameter budget, networks trained with knowledge distillation usually outperform models directly trained on the task. Sanh et al. [99], Tsai et al. [120], and Jiao et al. [54] applied different knowledge distillation schemes on the original BERT [24] to obtain lighter and faster models called DistilBERT, MiniBERT, and TinyBERT, respectively. Table 2 reports their compression, speed-up, and performance. Although knowledge distillation achieves impressive compression ratios and performance trade-offs, a large teacher model still needs to be trained, and the student may perform significantly worse than the teacher. For instance, $\text{BERT}_{\text{BASE}}$ achieves an accuracy of 52.8% on the CoLA task [129], while DistilBERT and TinyBERT only achieve 32.8% and 44.1%, respectively, according to Jiao et al. [54].

Table 2. Multiple knowledge distillations of $\text{BERT}_{\text{BASE}}$. Speed-ups are evaluated on GPUs.

| Model | Compression | Speed-up | Mean Relative Performance |
|---|---|---|---|
| $\text{BERT}_{\text{BASE}}$ [24] | 1.0× | 1.0× | 100% |
| DistilBERT [99] | 1.7× | 1.6× | 97% |
| MiniBERT [120] | 6.0× | 2.6 − 4.3× | 97 − 99% |
| TinyBERT [54] | 7.5× | 9.4× | 97% |

**Mixed-Precision** [80]: Modern GPUs and TPUs perform at least twice as many half-precision (16 bits) float operations as single-precision (32 bits) ones. A popular approach to accelerate training and reduce memory consumption is storing and computing the weights, activations, and gradients in half-precision. A master copy of the weights is stored in single-precision for numerical stability and minimal performance loss. Thanks to NVIDIA's Automatic Mixed-Precision included in some of the most popular deep learning libraries, namely TensorFlow, PyTorch, and MXNet, using mixed precision can be as simple as adding one line of code. Consequently, we highly recommend mixed-precision. Jacob et al. [51] improved over this approach by quantizing both weights and activations as 8-bit integers and biases as 32-bit integers, effectively allowing inference to be performed using

integer-only arithmetic. Given a parameter matrix $\boldsymbol{W}$, $N$-bit quantization rounds each parameter to one of $2^N$ codewords corresponding to bins evenly spaced by a scale factor $s$ and shifted by a bias $z$ computed as follows:

$$s = \frac{\max \boldsymbol{W} - \min \boldsymbol{W}}{2^N - 1} \quad \text{and} \quad z = \text{round}\left(\frac{\min \boldsymbol{W}}{s}\right) \tag{10}$$

Each parameter $W_{i,j}$ is quantized to its nearest codeword, and dequantized as:

$$\hat{W}_{i,j} = \left(\text{round}\left(\frac{W_{i,j}}{s} + z\right) - z\right) \times s \tag{11}$$

In order to mitigate the performance loss associated with the low-precision approximation, Quantization Aware Training (QAT) [51] quantizes the parameters during training. Since quantization is not differentiable, gradients are approximated with a straight-through approximator [7]. Notably, Zafrir et al. [138] quantized all matrix product operations in BERT fully connected and embedding layers during training, reducing the memory footprint by 4× while retaining 99% of the original accuracy on the GLUE [124] and SQuAD [94] tasks. Stock et al. [107] achieved an even higher compression ratio with iterative product quantization (iPQ), which replaces vectors of weights by their assigned centroid, and quantization of those centroids. The authors reduced the size of a 16-layer Transformer by 25×, making the model only 14 MB, while retaining 87% of the original performance on the Wikitext-103 [78] benchmark.

While pruning and knowledge distillation achieve faster and lighter models by reducing the number of parameters, mixed-precision and quantization instead reduce the number of bits per parameter.

**Micro-Batching** [48]: Increasing model capacity and data throughput are efficient strategies for improving performances in deep learning. However, increasing data throughput requires transferring large mini-batches to the accelerators' memory[5], which is also used to store the model. One way to partially avoid the trade-off between mini-batch size and model size is to use model parallelism. GPipe [48] is a model parallelism library that enables users to distribute a model by grouping layers into cells assigned to accelerators. To avoid the communication bottleneck between accelerators due to the forward and backward operations, the authors proposed a novel batch-splitting algorithm that further splits the mini-batch into micro-batches. As soon as the first accelerator finishes the forward operation of the layers assigned to it for a micro-batch, it sends the result over the communication link and starts processing the next micro-batch. After finishing the last micro-batch's forward operation, the accelerators wait for the first micro-batch's backwards operation results. This waiting time can be used to recompute the forward operation and further reduce memory usage, known as rematerialization. Finally, once the backward operation is completed on the last micro-batch, the algorithm sums all micro-batch's gradients to obtain the mini-batch's gradient (see Figure 6). However, the result is not exact with layers that compute statistics across all mini-batch examples, such as a batch normalization layer [50]. Finally, GPipe is compatible with data parallelism, where multiple mini-batches are processed in parallel.

Huang et al. [48] empirically demonstrated that GPipe allows the maximum Transformer size to scale linearly with the number of accelerators. For instance, a TPU v3 with 16Gb of memory can only fit a 3-layer Transformer. With GPipe, the same TPU is able to fit 13 layers, while 128 TPUs are able to fit 1663 layers, which is 127.9× more. Additionally, the authors distributed a 48-layer Transformer across 8 TPUs and reported that the training throughput was 4.8 times higher with 32 micro-batches than with a single one.

[5] An accelerator denotes any device that accelerates computation, such as a graphics or tensor processing unit.

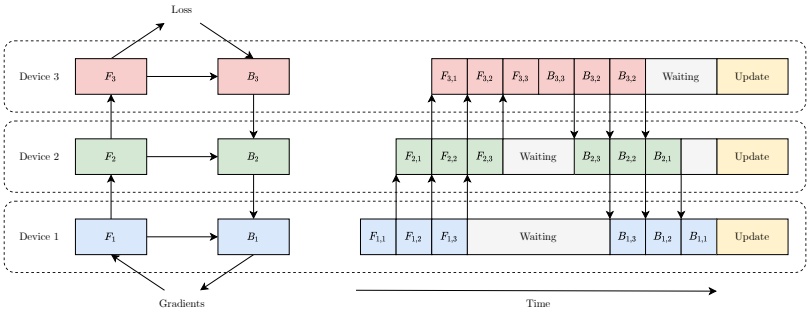


Fig. 6. Micro-Batching applied to a model distributed across three devices [48]. $F_i$ and $B_i$ denotes the sequential forward and backward operations, respectively, performed by the $i$-th device. Computation on a device may start as soon as the previous device in the computational graph has processed the first micro-batch. Therefore, micro-batching reduces the waiting time of each device at the cost of inter-device communications. Note that the model update is done synchronously at the end.

**Mixture of Experts** [52]: The core idea is to train multiple networks called experts, each of which specializes only in a subset of the data, and a manager or router, which forwards the input to the corresponding experts. A single network is used in practice, whose layers are composed of multiple subsets of parameters (experts), effectively resulting in a sparsely activated model as illustrated in Figure 7. Increasing the number of experts keeps the computational cost constant since the model always selects the same number of experts for each input regardless of the number of experts. Therefore, the mixture of experts (MoE) approach allows for massive models and is particularly efficient for distributed systems in which experts are spread across devices. In that case, the number of experts, and therefore parameters, scales with the number of devices available. Despite these advantages, the mixture of experts has not yet been widely adopted as the method is complex to deploy in practice. It imposes a communication cost between the devices, a computation cost to select the experts for each input position, and makes training unstable. Recently, Fedus et al. [30] introduced the Switch Transformer based on a carefully crafted mixture of experts. Notably, given a fixed amount of computation per input position, the Switch Transformer reached the same quality threshold as a vanilla Transformer five times faster (wall-clock time) on average. Additionally, when trained further, the Switch Transformer outperformed the vanilla baseline. However, this approach assumes that multiple regimes with distinct input to output relations produce the data.

Difficult tasks often require large models to achieve the desired performance. However, such models require powerful and expensive accelerators. Both micro-batching and the mixture of experts offer an alternative to train such models on many relatively weak and inexpensive GPUs at the cost of complex implementation.

**Sample-Efficient Objective** [19]: Large neural networks, especially Transformers, benefit from being pre-trained with an unsupervised objective before being fine-tuned on the task of interest, also called the downstream task. The core idea is to leverage large unlabelled datasets that are easy to automatically collect in order to learn the data underlying explanatory factors and ultimately improve the model performance. Concretely, pre-training initializes the network's weights in a "good" region of space. As pre-training of large models is often more compute-intensive than fine-tuning, researchers regularly share pre-trained models to facilitate their adoption. Most notably, Hugging Face [132] is an open-source library that contains an extensive collection of pre-trained

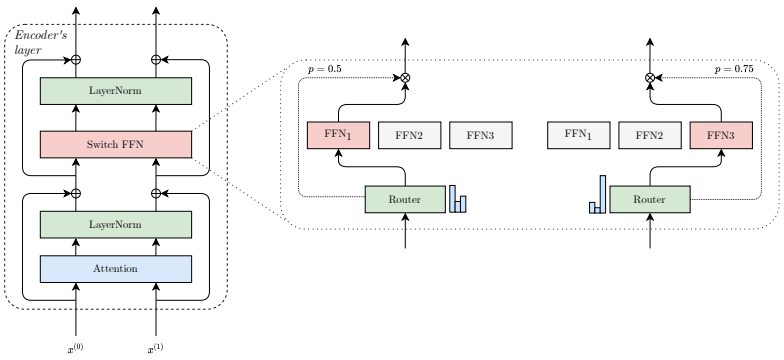


Fig. 7. The computational graph of a single layer of the Switch Transformer's encoder [30]. The Transformer's feed-forward network (FFN) has been replaced by a Switch FFN which independently routes each position to an expert. The expert's output is multiplied by the gate value. Note that the computational cost is independent of the number of experts since a single expert is active for each position.

Transformers under a unified API. Nonetheless, researchers must sometimes pre-train models themselves due to the peculiar nature of the data or the problem at hand. In that case, a sample-efficient objective will reduce the computation required.

Recently, Devlin et al. [24] popularized the Cloze procedure [117] for pre-training under the name of masked language model (MLM), which independently estimates the probability of masked words given the rest of the sequence. Practically, 15% of the words are randomly selected, of which 80% are masked, 10% are replaced by a random word, and 10% are left unchanged. This task is analogous to the reconstruction of corrupted input. Figure 8 illustrates the masked language model objective.

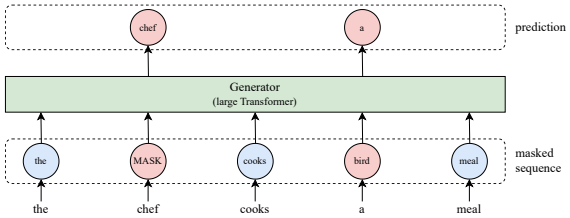


Fig. 8. The masked language model objective [24]. The masked words are depicted in red. The model makes a prediction only for the masked words; thus, MLM is computationally inefficient.

Clark et al. [19] introduced the replaced token detection objective to speed up pre-training; a small network (generator) first generates a plausible alternative for each masked word, then the large model (discriminator) predicts whether each word has been replaced (see Figure 9). While the masked language model makes a prediction only for the masked works, the replaced token detection makes a prediction for every word. Therefore, the latter is more computationally efficient than the former; in other words, less pre-training computations are required to achieve the same performance on downstream tasks. Additionally, the authors reported that the representations learned with their objective outperformed those learned with MLM given the same model size, data, and computation. Most notably, they were able to outperform GPT on the GLUE benchmark with 30× fewer computations.

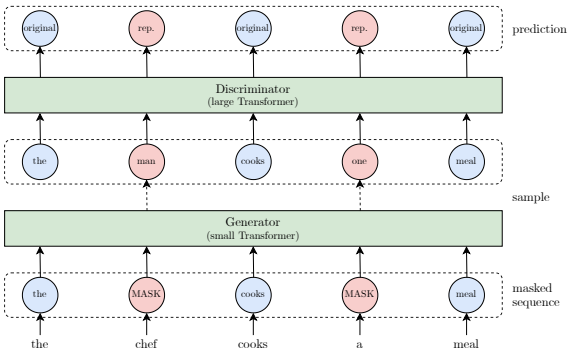


Fig. 9. The replaced token detection objective [19]. A plausible alternative of each masked word is sampled from a small generator network. Then a discriminator predicts whether each word has been replaced.

**Parameter Initialization Strategies**: Optimizing deep networks is challenging in part because of the considerable influence of the initial point on the iterative process. Notably, the initial point determines whether the algorithms converge at all and, if it does converge, the speed at which it converges as well as the quality of the solution [36]. Transformers are notoriously difficult to train, typically requiring carefully tuned optimizers with adaptive learning rates, learning rate schedulers, and large batches. Even then, convergence is not guaranteed. Consequently, Liu et al. [73] and Huang et al. [47] concurrently proposed initialization schemes for the Transformer that promise a smoother and faster optimization as well as better generalization performances.

Liu et al. [73] identified an amplification effect that significantly influences training: each layer heavily depends on its residual branch[6], making the optimization unstable as it amplifies small parameter perturbations. Ultimately, the amplification effect may produce a notable change in the Transformer's output. Nonetheless, the authors observed that heavy dependencies on the residual branches are necessary to unlock the Transformer's potential and achieve better results. In order to mitigate the amplification effect, Liu et al. [73] introduced the Adaptive Model Initialization strategy, or Admin, that controls the dependency on the residual connections in the early stage of training with a new parameter $\boldsymbol{\omega}$. Formally, the $i$-th sub-layer output is given by

$$\boldsymbol{X}_i = \text{LayerNorm}(f_i(\boldsymbol{X}_{i-1}) + \boldsymbol{X}_{i-1} \odot \boldsymbol{\omega}_i), \tag{12}$$

where $f_i(\boldsymbol{X})$, $\boldsymbol{X}_{i-1}$, and $\boldsymbol{X}_i$, denote the function, input, and output of the $i$-th sub-layer, respectively. Although this is equivalent to rescaling some model parameters, the authors observed that rescaling leads to unstable training in half-precision.

The proposed initialization strategy requires three steps. First, the model parameters are initialized with a standard method such as the Xavier initialization [34] and the Admin parameter $\boldsymbol{\omega}$ with ones. Then, one or a small number of mini-batches are forward propagated without updating the parameters and record the output variance of each residual branch $\text{Var}[f_i(\boldsymbol{X}_{i-1})]$. Finally, the Admin parameter is initialized as $\boldsymbol{\omega}_i = \sqrt{\sum_{j<i} \text{Var}[f_j(\boldsymbol{X}_{j-1})]}$. Once the model has been trained, $\boldsymbol{\omega}$ may be discarded.

The amplification effect is, however, not the only mechanism that makes Transformers notoriously difficult to train. Huang et al. [47] addressed two other issues: (i) Transformers are typically trained with optimizers that rely on adaptive learning rates as conventional SGD fails to train them

[6]For a residual block $f(x) + x$, the residual branch refers to $f(x)$ and the skip connection, shortcut connection, or residual connection refers to $x$.

effectively. However, adaptive learning rates have a problematically large variance in the early stages of optimization, resulting in convergence issues [72]; and (ii) the magnitude of the error signal propagated through LayerNorm is inversely proportional to the magnitude of the input [135]. Specifically, the norm of the layer normalization gradient is proportional to:

$$\left\| \frac{\partial \text{LayerNorm}(\boldsymbol{x})}{\partial \boldsymbol{x}} \right\| = O\left( \frac{\sqrt{d}}{\|\boldsymbol{x}\|} \right) \tag{13}$$

Consequently, if the input norm $\|\boldsymbol{x}\|$ is larger than $\sqrt{d}$, backpropagating through layer normalization reduces the gradient magnitude for layers closer to the input. As a solution to both problems, Huang et al. [47] proposed an initialization strategy called T-Fixup that restricts the magnitude of the updates in the early stages of training, thus mitigating the vanishing gradient issue while eliminating the need for layer normalization and warmup.

While Liu et al. [73] and Huang et al. [47] claim faster convergence, they did not report the improvement.

**Architecture Search**: One of the most challenging goals in deep learning is to automatically design networks. Indeed, the problem of finding architectures that achieve the best performance with the fewest operations and lowest memory footprint in a discrete search space is an NP-hard combinatorial optimization problem. Over the years, multiple approaches to Neural Architecture Search (NAS) have been proposed, including reinforcement learning [141], evolutionary algorithms [95], and bilevel optimization [71]. Notably, Zoph et al. [142] demonstrated that NAS is able to surpass human-designed architectures on ImageNet by 1.2% top-1 accuracy while using 28% fewer computations. Nonetheless, neural architecture search methods are computationally expensive as they usually require training each candidate model from scratch. As a solution, Pham et al. [88] proposed Efficient NAS (ENAS), which constrains all candidates to be subgraphs of a single computational graph, that is, to share parameters. Therefore, the ENAS's controller decides which operations are activated and relies on the models' ability to adapt, similarly to dropout [106]. Efficient NAS reduces the search computational budget by 1,000× over the original NAS [141]. Alternatively, Liu et al. [71] proposed the Differentiable Architecture Search (DARTS), which casts the NAS problem as a differentiable bilevel optimization problem. The first level consists of a continuous relaxation of the discrete search space using a Softmax function over a list of candidate operations, and the second level involves the model's weights. However, the bilevel formulation requires training the weights to convergence to evaluate the architecture gradient. To avoid this substantial cost, the authors made the approximation of taking a single gradient step of the weights for one gradient step of the architecture parameters. The authors obtained comparable performances to non-differentiable NAS methods on ImageNet in the mobile setting using only 4 GPU-days, compared to 3,150 for evolutionary algorithms [95] and 2,000 for NAS [142]. Differentiable Architecture Search obtained comparable results to ENAS with a similar computational budget. We refer the reader to Elsken et al. [29] survey for further detail on architecture search methods.

Nevertheless, neural architecture search methods are challenging to apply on Transformers due to the memory requirements and training time. Therefore, recent works introduced methods better suited for the Transformer. So et al. [105] modified the tournament selection evolutionary architecture search [95] with Progressive Dynamic Hurdles (PDH), which dynamically allocates resources to more promising architectures according to their performances. With PDH, the authors optimized transformer architectures directly on the WMT'14 En-De task [9] which requires 10 hours of computation on a Google TPU v2 for the base Transformer model. Training directly on this dataset is essential since the authors did not find a smaller surrogate dataset that transfers well, such as CIFAR-10 for ImageNet. The Evolved Transformer matched the vanilla Transformer's

performance with only 78% of its parameters. Recently, Tsai et al. [119] profiled the Transformer's components on a TPU v2 and observed that some mechanisms substantially impact inference time: attention queries, keys, and values dimensions, width and depth of feed-forward layers, number of attention heads, and layer normalization mean computation. By decomposing these components into building blocks and using binary variables, the authors perform a one-shot search for both the architecture and the parameters with a single loss. They optimized this loss with gradient descent on a continuous relaxation of the binary variables and used policy gradient algorithm. Tsai et al. [119] were able to make miniBERT 1.7× faster with a performance drop smaller than 0.3%. Compared to the original BERT, this is 33 to 36× faster.

Neural architecture search is a promising tool to design lighter and faster Transformers automatically. Nonetheless, NAS imposes a high computational and memory cost, which may be avoided by carefully engineering the architecture instead. For instance, the Lite Transformer [133] leverages the Long-Short Range Attention (LSRA), where a convolutional layer is applied in parallel to the self-attention in order to learn the local dependencies separately. The carefully handcrafted Lite Transformer outperforms the Evolved Transformer [105] for the mobile NLP setting while requiring about 14,000× less GPU time.

**Conditional Computing** [6]: Although large models are necessary for hard examples, smaller models are likely to perform as well, if not better, on simpler ones. For instance, many words such as "car" are easy to translate, while a few such as "can" require careful consideration of the context[7]. As of this survey's writing, most architectures apply a fixed number of operations to all examples regardless of their difficulty. A more efficient approach would be to reduce the amount of computation for simple examples. As a solution, Bengio [6] introduced conditional computing, which dynamically adapts the model's computational graph as a function of the input.

One way to implement conditional computing is with a mixture of experts, as introduced previously. In that case, only a subset of the parameters is used for a given input, making the computational graph sparse and the computation time almost constant with respect to the model size. Another approach consists of keeping the number of parameters constant and letting the model adjust its computation time separately for each input (according to the input's value). This approach is called Adaptive Computation Time (ACT) [38] and uses a recurrent mechanism to transform the representations until a halting probability exceeds a given threshold. The model learns to control this probability to minimize both the prediction error and the number of iterations, called the *ponder cost*, which prevents the model from using an infinite amount of computation before making a prediction. One shortcoming of the Adaptive Computation Time is its sensitivity to the ponder cost, which controls the trade-off between speed and accuracy.

Dehghani et al. [23] applied ACT to a Transformer with a recurrent mechanism for the architecture's depth. To implement this mechanism, the authors defined encoder and decoder blocks similar to the original Transformer, except that each block is recurrent, sending its output back as its input until the ponder cost becomes too high. Note that a fixed number of recurrent steps is equivalent to a Transformer with tied parameters across all layers. With this new architecture called Universal Transformer, the authors claimed that it is computationally universal (Turing-complete) given enough memory. This property may help Transformers generalize to sequences longer than the ones seen during training. The authors obtained state-of-the-art results on algorithmic and language understanding tasks. ACT and the Universal Transformer apply the same layers iteratively, which may not be sufficiently flexible. Elbayad et al. [28] addressed this limitation with the Depth-Adaptive Transformer (DAT), which applies different layers at every depth. The DAT

[7]Depending on the context, the word "can" has various meanings, including "be able to", "may", "jail", and "metal container". See https://www.wordreference.com/definition/can.

matches the performance of a well-tuned Transformer baseline while reducing the computation by up to 76%. However, the authors did not provide a comparison between the Universal Transformer and DAT.

In the same way that complex examples may require more computations, some may require access to a longer context. As a solution, Sukhbaatar et al. [111] dynamically adjusted the attention span, that is, the context length, by learning to mask the compatibility scores depending on the input. Their approach achieved state-of-the-art on `text8` and `enwik8` [77] while requiring significantly fewer computations. Alternatively, Li et al. [65] introduced the Decoder-end Adaptive Computation Steps (DACS), which monotonically computes halting probabilities along with the encoder states and stops the decoder computations in order to produce an output when the accumulation of probabilities exceeds a given threshold. In other words, each decoder step only looks at the necessary information as measured by the halting probabilities instead of looking at the entire input sequence.

## 4 SPECIALIZED APPROACHES

Since the Transformer's quadratic complexity comes from the attention mechanism, most specialized methods rely on a fast and light approximation of the original full attention. As will be explained in greater detail in the rest of this section, the attention weight matrix is dominated by a few large values and is approximately low-rank. These observations justify two distinct lines of work: sparse attention and factorized attention. Alternatively, the complexity may be reduced without altering the original attention mechanism and thus the Transformer's capacity by directly modifying the network's architecture. Let us first investigate the approaches that rely on sparse attention.

Note that some approaches only consider autoregressive tasks, such as the left-to-right language model, and in that case, the connectivity matrix is lower triangular as it is not permitted to attend to future positions. Whenever possible, such works have been extended to the more general case where attending to future positions is allowed in order to ease the comparison between the different approaches.

### 4.1 Sparse Attention

Due to the exponential nature of the Softmax, only a few positions are strongly attended to. Consequently, a conceptually simple way of reducing the Transformer's complexity is to make the matrix $\boldsymbol{QK}^\top$ sparse[8], in other words, to only allow each position to attend to a subset of the positions. Let us investigate sparse patterns that are (i) fixed and random, (ii) learned and adaptive, and (iii) identified with clustering and locality sensitive hashing.

**Fixed and Random Sparse Patterns** [5, 16, 41, 66, 90, 125, 139]: One of the first models to consider fixed sparse patterns is the Star-Transformer introduced by Guo et al. [41], which reduced the complexity from quadratic to linear by only allowing attention between adjacent positions. In order to preserve the Transformer's ability to model long-term dependency, the authors relied on a single global token. Global tokens, also known as shared relay nodes, can attend to every position, and every position can attend to global tokens. Let us assume that the global token is located at position 0. The $i$-th output position is allowed to attend to every input position if $i = 0$, otherwise, it is allowed to attend to the $j$-th input positions for $j = 0$ and if $i - 1 \leq j \leq i + 1$. Figure 10 illustrates the Star-Transformer attention pattern.

Concurrently, Child et al. [16] introduced the Sparse Transformer which reduced the complexity to $O(n\sqrt{n})$ with two different sparse attention patterns: strided and fixed. Strided attention allows the $i$-th output position to attend to the $j$-th input position if one of the two following conditions

[8]Since the matrix $\boldsymbol{QK}^\top$ is passed through a Softmax function, the masked values are set to minus infinity, effectively setting their contribution to $e^{-\infty} = 0$.

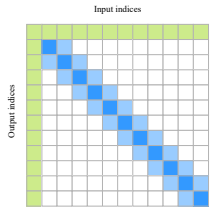


Fig. 10. The connectivity matrices of the Star-Transformer [41].

is satisfied: $(i+s) > j > (i-s)$ or $(i-j) \bmod s = 0$, where the stride $s$ is chosen to be close to $\sqrt{n}$. Similarly, fixed attention allows $i$ to attend to $j$ if one of the two following conditions is satisfied: $\text{floor}(j/s) = \text{floor}(i/s)$ or $(j \bmod s) \geq (s-c)$, where $c$ is an hyperparameter. Figure 11 illustrates the strided and fixed attention patterns.

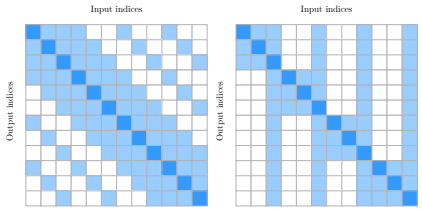


Fig. 11. The connectivity matrices of the Sparse Transformer Child et al. [16]. (Left) Strided attention with a stride of 3. (Right) Fixed attention with a stride of 3 and $c = 1$.

Alternatively, Wang et al. [125] introduced the Cascade Transformer, which relies on sliding window attention whose size grows exponentially with the number of layers. More specifically, the number of cascade connections at the layer $l$ is equal to $2.b.m^l - 1$, where $b$ is the base window size and $m$ is the cardinal number; therefore reducing the complexity to $O(n.b.m^l)$. Cascade attention is well suited for shallow networks, but its complexity tends to that of the full attention in deep networks as depicted by the connectivity matrices in Figure 12.

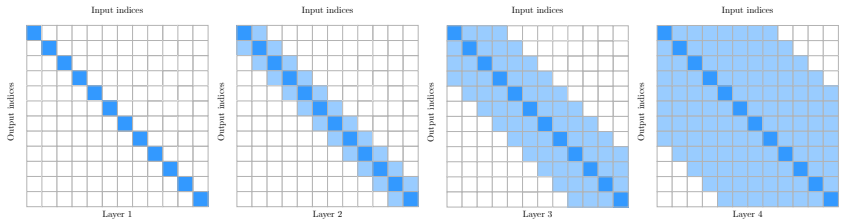


Fig. 12. The connectivity matrices of the Cascade attention [125] for the first four layers with a base window $b = 1$ and a cardinal number $m = 2$. For instance, the window size of the third layer ($l = 2$) is equal to $2 \times b \times m^l - 1 = 7$.

Li et al. [66] introduced the LogSparse-Transformer for forecasting fine-grained time series with strong long-term dependencies. The LogSparse-Transformer relies on the eponym attention that allows the $i$-th output to attend to the $j$-th inputs for $j \in \{-2^{\lfloor \log_2 i \rfloor}, i - 2^{\lfloor \log_2 i \rfloor - 1}, \ldots, i - 2^1, i - 2^0, i, i + 2^0, i + 2^1, \ldots, i + 2^{\lfloor \log_2 (n-i) \rfloor - 1}, i + 2^{\lfloor \log_2 (n-i) \rfloor}\}$ where $\lfloor . \rfloor$ denotes the floor operation and

$N$ denotes the sequence length. Figure 13 illustrates the connectivity matrix of the LogSparse attention. Since only $O(\log n)$ positions are attended to by each of the $n$ positions, the complexity of the LogSparse attention is $O(n \log n)$. Additionally, the authors proposed two alternatives: (1) to allow the $i$-th output to attend to the first $k$ input positions, after which the LogSparse attention is resumed, and (2) to divide the input sequence into subsequences, and to apply the LogSparse attention on each of them.

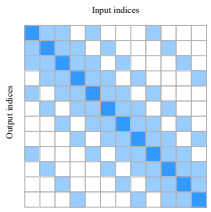


Fig. 13. The connectivity matrix of the LogSparse attention Li et al. [66].

Qiu et al. [90] introduced BlockBERT, which relies on the block-wise attention: the input sequence is split into $n_b$ non-overlapping blocks, and positions in block $i$ are only allowed to attend to positions in block $\pi(i)$, where $\pi$ denotes a permutation. The author chose to generate the permutations by simply shifting the positions. For instance, the possible permutations of $\{1, 2, 3\}$ are $\{1, 2, 3\}$, $\{3, 1, 2\}$, and $\{2, 3, 1\}$. The permutation $\{2, 3, 1\}$ means that the first block attends to the second block, the second block attends to the third block, and the third block attends to the first block. In the multi-head setting, a different permutation[9] is assigned to each head. More formally, the output position $i$ is only allowed to attend to input $j$ if the following condition is satisfied:

$$\pi\left(\left\lfloor \frac{(i-1)n_b}{n} + 1 \right\rfloor\right) = \left\lfloor \frac{(j-1)n_b}{n} + 1 \right\rfloor \tag{14}$$

Figure 14 illustrates the connectivity matrix of the block-wise attention where a sequence of length $n = 12$ is split into $n_b = 3$ blocks. Although the block-wise attention reduces the memory and computational cost by a factor $n_b$, the complexity remains quadratic with respect to the sequence length.

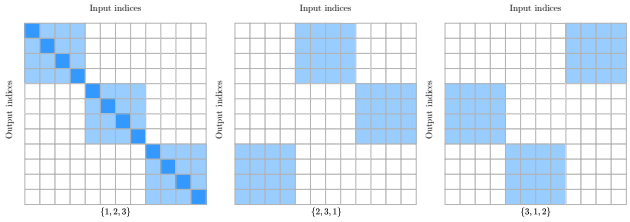


Fig. 14. The connectivity matrices of the block-wise attention [90] for $n_b = 3$ blocks. The corresponding permutations are written below the connectivity matrices.

Beltagy et al. [5] introduced the Longformer which further reduces the complexity to $O(n)$ using a combination of sliding window and global attentions (see Figure 15). The assumption behind the sliding window attention is that the most useful information is located in each position's

[9]Note that if the number of heads is greater than the number of permutations, multiple heads must be assigned the same permutation.

neighbourhood. The sliding window attention is limited in that it requires $O(\sqrt{n})$ layers to model long-range dependencies. Thus, a few preselected tokens have a global attention: they can attend to every position and be attended by every position. Consequently, the maximum path length between any two positions is equal to 2. Zaheer et al. [139] introduced BigBird, which also achieves a linear complexity using a combination of random, sliding window, and global attentions (see Figure 15). BigBird has two configurations that the authors referred to as internal transformer construction (ITC) and extended transformer construction (ETC). Similarly to the Longformer, the former uses existing positions for global attention, while the latter uses additional tokens, increasing the model's capacity and performance. Interestingly, the extra location of ETC may be seen as a form of memory. The authors proved that their sparse factorization preserves the theoretical properties of Transformers with the full attention: the model is both a universal approximator of sequence functions and Turing complete. However, BigBird without random attention outperformed BigBird with it in most of their experiments.

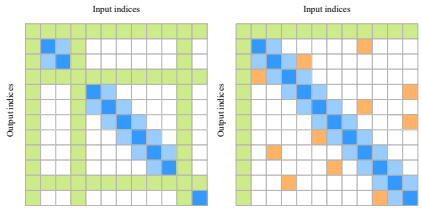


Fig. 15. The connectivity matrices of two sparse attention schemes. (Left) Longformer [5]. (Right) BigBird [139]. The attention is the combination of sliding window attention (blue), global attention (green), and random attention (orange).

**Learned and Adaptive Sparse Patterns** [20, 104, 114]: Fixed and random patterns are hand-crafted and may not be suitable for the data and task at hand. One may instead learn the relevant patterns and adapt them based on the content.

In order to increase the flexibility of the block-wise attention, Tay et al. [114] introduced the sparse Sinkhorn attention, which is equivalent to the block-wise attention whose keys have been sorted in a block-wise fashion. In other words, the permutations are learned. More specifically, the sparse Sinkhorn attention transforms the input sequence $\boldsymbol{X} \in \mathbb{R}^{n \times d}$ into $\boldsymbol{X}' \in \mathbb{R}^{n_b \times d}$ where $n_b$ is the number of blocks, and where $\boldsymbol{X}'_i$ is equal to the sum of the input in that block. A simple feed-forward network then learns a mapping $\boldsymbol{R}_i \in \mathbb{R}^{n_b}$ from the $i$-th block $\boldsymbol{X}'_i$ to all blocks. In order to obtain a sorting matrix from $\boldsymbol{R} \in \mathbb{R}^{n_b \times n_b}$, that is, a matrix comprising only 0s and 1s, and whose rows and column sum to one, the rows and columns are iteratively normalized. The sorting matrix is then used to permute the keys, effectively learning which block to attend (see Figure 16). The sparse Sinkhorn attention reduces the complexity to $O(n_b^2)$. Nonetheless, since the block size is constant in the original paper, the complexity remains quadratic with respect to the sequence length. Additionally, the authors proposed a truncated version of the sparse Sinkhorn attention, which selects a few keys after sorting them, further reducing the complexity to $O(n)$.

Recently, Shi et al. [104] put under the microscope the attention patterns learned by BERT [24] and observed that the diagonal elements are less important compared to other positions, that is, they contribute the least to the output, while neighbourhood positions and special tokens are prominent. To confirm their observations, they dropped the diagonal element in BERT's attention such that each position is not allowed to attend to itself and noted that the performance remains comparable to the original model. Additionally, they observed that models for different tasks have various degrees of redundancy and hence can achieve various sparsity levels before significantly

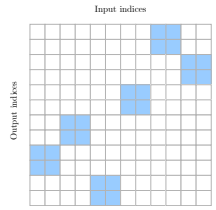


Fig. 16. The connectivity matrix of the sparse Sinkorn attention [114].

dropping performance. Consequently, Shi et al. [104] proposed to learn sparsity patterns for each task in an end-to-end fashion with the Differentiable Attention Mask (DAM) algorithm. Let us denote the attention score between the $i$-th output position (query) and $j$-th input position (key) as $\alpha_{i,j}$. They proposed to compute the attention mask $M_{i,j}$ as the Gumbel-Sigmoid [76] of the attention score $\alpha_{i,j}$:

$$M_{i,j} = \text{Gumbel-Sigmoid}(\alpha_{i,j}) = \text{Sigmoid}\left(\frac{\alpha_{i,j} + G_1 - G_2}{\tau}\right) \tag{15}$$

where $G_1, G_2$ are independent Gumbel noises $G_k = -\log(-\log(U_k))$ generated from a uniform distribution $U_k \sim \mathcal{U}(0, 1)$, and where $\tau$ is a temperature hyperparameter. Note that the Gumbel-Sigmoid becomes binary as $\tau$ approaches 0. A penalty term $\lambda\|M\|_1$ is added to the loss to control the trade-off between performance and sparsity. The resulting model called SparseBERT achieved 91.2% sparsity while maintaining an average score of 80.9% on GLUE, i.e., only 3% lower than the full BERT. Such an approach deviates from previous sparse attention whose patterns have been manually handcrafted. To avoid learning completely unstructured sparsity patterns, the authors proposed to enforce the first and last row/column of the attention mask to be active and all positions on each line parallel to the diagonal to share their mask parameters.

As mentioned above, due to the exponential nature of the Softmax, most positions are lightly attended to. In other words, most attention weights are small but non-zero. Instead, Correia et al. [20] introduced the Adaptively Sparse Transformer that replaces the Softmax by the $\alpha$-entmax function, a differentiable generalization of the Softmax that pushes small weights to be exactly zero. Formally, the $\alpha$-entmax function is defined as:

$$\alpha\text{-entmax}(\boldsymbol{z}) = \underset{\boldsymbol{p} \in \Delta^d}{\text{argmax}}\ \boldsymbol{p}^\top \boldsymbol{z} + \boldsymbol{H}_\alpha^T(\boldsymbol{p}), \tag{16}$$

where $\Delta^d = \{\boldsymbol{p} \in \mathbb{R}^d : \sum_i p_i = 1\}$ and, for $\alpha \geq 1$, $\boldsymbol{H}_\alpha^T$ is the Tsallis continuous family of entropies:

$$\boldsymbol{H}_\alpha^T(\boldsymbol{p}) = \begin{cases} \frac{1}{\alpha(\alpha-1)} \sum_j (p_j - p_j^\alpha), & \alpha \neq 1 \\ -\sum_j p_j \log p_j, & \alpha = 1. \end{cases} \tag{17}$$

The authors showed that the solution to the equation 16 is

$$\alpha\text{-entmax}(\boldsymbol{z}) = [(\alpha - 1)\boldsymbol{z} - \lambda \mathbf{1}]_+^{\frac{1}{\alpha-1}}, \tag{18}$$

where $[]_+$ denotes the ReLU function, $\mathbf{1}$ denotes the vector of ones, and $\lambda$ is the Lagrange multiplier corresponding to the $\sum_i p_i = 1$ constraint.

Interestingly, when $\alpha = 1$, the $\alpha$-entmax is equivalent to the Softmax, and the attention is dense, and when $\alpha > 1$, the output is permitted to be sparse. In their experiments, a scalar parameter $a_{i,j}$ is learned for the $j$-th attention head of the $i$-th layer, and $\alpha_{i,j}$ is computed as:

$$\alpha_{i,j} = 1 + \text{sigmoid}(a_{i,j}) \in ]1, 2[ \tag{19}$$

Nonetheless, the Adaptively Sparse Transformer computes the attention score for each pair of queries and keys. Consequently, the sparsity cannot be leveraged to improve the memory and computation, resulting in a model that is 25% slower than the original Transformer in terms of tokens per second.

As of this survey's writing, unstructured sparse attention (whether fixed, random or learned) does not benefit from efficient implementations and therefore cannot result in memory and computational improvements. Nonetheless, there are exciting researches in that direction, as noted by Hooker [45]. In contrast, some structured sparsity patterns benefit from efficient implementations. Recently, NVIDIA introduced its Ampere architecture which efficiently compresses 2:4 structured sparsity on rows, that is, two non-zero values in every four entries.

**Clustering and Locality-Sensitive Hashing** [59, 96]: The Softmax function is dominated by the largest values, that is, by the keys and queries that have the largest dot product. Therefore, the attention may be approximated by only comparing the most similar keys and queries. Although this approach is a form of adaptive sparsity as the patterns depend on the data, they are presented separately due to their conceptual difference.

Kitaev et al. [59] introduced the Reformer, which selects the set of keys that the query can attend to by grouping them with an angular multi-round locality-sensitive hashing (LSH). Such hashing scheme has a high probability of assigning the same value to similar vectors. Formally, queries and keys are shared ($Q = K$) and bucketed using $b$ hash values obtained as follows:

$$\boldsymbol{p} = [\boldsymbol{x}^\top \boldsymbol{R}; -\boldsymbol{x}^\top \boldsymbol{R}] \tag{20}$$

$$h(\boldsymbol{x}) = \underset{i}{\operatorname{argmax}}(p_i) \tag{21}$$

where ; denotes the concatenation operation, and where $\boldsymbol{x} \in \mathbb{R}^d$ is a query/key and $\boldsymbol{R} \in \mathbb{R}^{d \times b/2}$ is a random rotation matrix. Output positions are only allowed to attend to input positions that are in the same bucket (see Figure 17). They are, however, not allowed to attend to themselves because the dot product of a vector with himself will almost always be greater than the dot product with other positions.

The authors chose a constant bucket size $l_B$, resulting in a number of buckets $n_B = n/l_B$. The attention complexity is $O(n_B \times l_B^2)$ which simplifies as $O(n)$. This does not take into account the computation of the hash values for each position. As only $\log n_B$ bits are required to encode $n_B$ buckets, the complexity of computing hash values is given by $O(n \log n_B)$, which simplifies as $O(n \log n)$. Consequently, the complexity of the Reformer's attention is $O(n \log n)$.

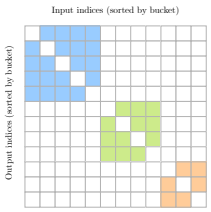


Fig. 17. The connectivity matrix of the Reformer [59]. Queries and keys are bucketed using LSH then sorted by their bucket. Therefore, the $i$-th row of the connectivity matrix may not correspond to the $i$-th position in the input sequence. Units can only attend other units in the same bucket, but not themselves because queries and keys are equal. The colour represents buckets.

The Maximum Inner Product Search (MIPS) problem is the task of searching for the vector $K_j$ in $K = \{K_1, K_2, \cdots, K_n\}$ that maximizes the dot product with a given vector $Q_i$. Note that the MIPS

problem is particularly useful for the attention mechanism as $Q_i^\top K_j$ is directly proportional to the contribution of the $j$-th value for the $i$-th attention's output. There are multiple approaches to approximately solve this problem, including tree-based and LSH-based. When the norm of every $K_j$ is constant, the problem is equivalent to the Nearest Neighbour Search (NNS). Motivated by this observation and to avoid the computational cost of learning sparsity patterns, Roy et al. [96] proposed the Routing Transformer that relies on an online mini-batch version of $k$-means and a set of centroids learned along the rest of the parameters. Like the Reformer, queries can only attend to keys from the same cluster, inducing an adaptive or content-based sparsity pattern.

### 4.2 Factorized Attention

Wang et al. [126] demonstrated that the attention matrix $\text{Softmax}\left(\boldsymbol{Q}\boldsymbol{K}^\top/\sqrt{d}\right)$ is approximately low rank. Consequently, another approach to reduce the Transformer's complexity is to approximate the attention by factorizing it into the product of two matrices with lower dimensions.

**Low-Rank Factorization** [113, 126, 136]: Wang et al. [126] introduced the Linformer, a linear complexity model that approximates the attention with a low-rank factorization by first projecting each key to a lower dimension before performing the dot product, thereby saving time and memory. Formally, the low-rank attention is given by:

$$\text{Attention}(\boldsymbol{X}) = \underbrace{\text{Softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d}}\right)}_{n\times n}\underbrace{\boldsymbol{V}}_{n\times d} \approx \underbrace{\text{Softmax}\left(\frac{\boldsymbol{Q}(\boldsymbol{E}\boldsymbol{K})^\top}{\sqrt{d}}\right)}_{n\times k}\underbrace{\boldsymbol{F}\boldsymbol{V}}_{k\times d} \tag{22}$$

where $\boldsymbol{E}, \boldsymbol{F} \in \mathbb{R}^{k\times n}$, with $k \ll n$, are two linear projection matrices learned during training. The authors showed that $\boldsymbol{E}$ and $\boldsymbol{F}$ could be shared across heads and layers with virtually no performance penalty.

Tay et al. [113] introduced a family of models called Synthesizers that learn the compatibility scores without computing the pairwise dot products between the queries and keys. For instance, the Dense Synthesizer learns the compatibility scores with a simple position-wise feed-forward network that projects each of the $n$ rows of $\boldsymbol{X}$ from $\mathbb{R}^{1\times d}$ to $\mathbb{R}^{1\times n}$:

$$\text{F}(\boldsymbol{X}_i) = \max(0, \boldsymbol{X}_i\boldsymbol{W}_1 + \boldsymbol{b}_1)\boldsymbol{W}_2 + \boldsymbol{b}_2 \tag{23}$$

where $\boldsymbol{W}_1 \in \mathbb{R}^{d\times d}$ and $\boldsymbol{W}_2 \in \mathbb{R}^{d\times n}$. Finally, the attention is given by:

$$\text{Attention}(\boldsymbol{X}) = \text{Softmax}(F(\boldsymbol{X}))G(\boldsymbol{X}) \tag{24}$$

where $G(\cdot) : \mathbb{R}^{n\times d} \to \mathbb{R}^{n\times d}$ is a projection of the input akin to the values. In order to improve the efficiency, the authors proposed the Factorized Dense Synthesizer which first project the input $\boldsymbol{X}$ with two feed-forward networks:

$$\boldsymbol{A} = F_A(\boldsymbol{X}) \in \mathbb{R}^{n\times a} \quad \text{and} \quad \boldsymbol{B} = F_B(\boldsymbol{X}) \in \mathbb{R}^{n\times b}, \tag{25}$$

such that $a\times b = n$. Then, two tiling functions $H_A(\cdot) : \mathbb{R}^{n\times a} \to \mathbb{R}^{n\times(a.b)}$ and $H_B(\cdot) : \mathbb{R}^{n\times b} \to \mathbb{R}^{n\times(b.a)}$ are applied to $\boldsymbol{A}$ and $\boldsymbol{B}$, respectively. Note that a tiling function simply repeats a vector multiple times. Finally, the attention of the Factorized Dense Synthesizer is given by:

$$\text{Attention}(\boldsymbol{X}) = \text{Softmax}(H_A(\boldsymbol{A})H_B(\boldsymbol{B})^\top)G(\boldsymbol{X}) \tag{26}$$

Additionally, the authors proposed a baseline called the Factorized Random Synthesizer, whose compatibility scores are independent of the input. Formally, the Factorized Random Synthesizer's attention is given by:

$$\text{Attention}(\boldsymbol{X}) = \text{Softmax}(\boldsymbol{R}_1\boldsymbol{R}_2^\top)G(\boldsymbol{X}) \tag{27}$$

where $\boldsymbol{R}_1, \boldsymbol{R}_2 \in \mathbb{R}^{n \times k}$ are two low-rank matrices learned during training. Although the Synthesizers eliminate the need to compute the pairwise dot products, which speed up the model in practice, the complexity remains quadratic with respect to the sequence length.

The Nyströmformer [136] relies on the Nyström method to generate a low-rank approximation of the Softmax matrix. However, applying the Nyström method directly to the Softmax would require to compute the $\boldsymbol{Q}\boldsymbol{K}^\top$ product, which requires $O(n^2)$ computations and memory. As a solution, the Nyströmformer creates two subsets $\tilde{\boldsymbol{K}}$ and $\tilde{\boldsymbol{Q}}$ of columns, called landmarks, from $\boldsymbol{K}$ and $\boldsymbol{Q}$, respectively. The authors applied the segment-means approach, which computes the landmarks as the averages over predefined spans of keys and queries. Let $\boldsymbol{S}_{AB}$ denotes $\text{Softmax}(\boldsymbol{A}\boldsymbol{B}^\top/\sqrt{d})$ for any matrix $\boldsymbol{A}$ and $\boldsymbol{B}$. The Nyströmformer approximates the Softmax matrix as:

$$\text{Softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d}}\right) \approx \boldsymbol{S}_{Q\tilde{K}}\boldsymbol{S}^{+}_{\tilde{Q}\tilde{K}}\boldsymbol{S}_{\tilde{Q}K} \tag{28}$$

where the superscript + denotes the Moore-Penrose inverse typically computed with the singular value decomposition (SVD). Since the SVD is inefficient on GPU, the authors relied on an iterative method that approximate $\boldsymbol{S}^{+}_{\tilde{Q}\tilde{K}}$ as $Z^+$. Finally, the Nyströmformer's attention is given by:

$$\text{Attention}(\boldsymbol{X}) \approx \boldsymbol{S}_{Q\tilde{K}}Z^{+}\boldsymbol{S}_{\tilde{Q}K}\boldsymbol{V} \tag{29}$$

which can be efficiently encoded in a computational graph.

Provided that the number of landmarks is constant and much smaller than the sequence length, the Nyströmformer complexity is $O(n)$. Depending on the number of landmarks and the sequence length, the authors reported substantial gains over the Linformer and Longformer on the masked language model and sentence order prediction objectives. Additionally, the representations learned by the Nyströmformer appear to transfer as well as BERT to different NLP tasks. Nonetheless, a more extensive evaluation of the Nyströmformer remains necessary.

**Kernel Attention** [18, 56]: A kernel $K(\cdot,\cdot)$ is a function that takes two vectors as arguments and returns the product of their projection by a feature map $\phi(\cdot)$:

$$K(\boldsymbol{x},\boldsymbol{y}) = \phi(\boldsymbol{x})^\top\phi(\boldsymbol{y}) \tag{30}$$

Katharopoulos et al. [56] interpreted the Softmax as a kernel, decomposed it as an inner product in the right space, and rearrange the computations in a clever way to reduce the complexity. More specifically, the self-attention of a given query $\boldsymbol{Q}_i$ may be rewritten using a mapping $\phi(\cdot)$:

$$\text{Softmax}\big(\boldsymbol{Q}_i^\top\boldsymbol{K}^\top\big)\boldsymbol{V} = \frac{\sum_{j=1}^{n}\exp\big(\boldsymbol{Q}_i^\top\boldsymbol{K}_j\big)\boldsymbol{V}_j}{\sum_{j=1}^{n}\exp\big(\boldsymbol{Q}_i^\top\boldsymbol{K}_j\big)} = \frac{\sum_{j=1}^{n}\phi\big(\boldsymbol{Q}_i\big)^\top\phi\big(\boldsymbol{K}_j\big)\boldsymbol{V}_j}{\sum_{j=1}^{n}\phi\big(\boldsymbol{Q}_i\big)^\top\phi\big(\boldsymbol{K}_j\big)} = \frac{\phi\big(\boldsymbol{Q}_i\big)^\top\sum_{j=1}^{n}\phi\big(\boldsymbol{K}_j\big)\boldsymbol{V}_j^\top}{\phi\big(\boldsymbol{Q}_i\big)^\top\sum_{j=1}^{n}\phi\big(\boldsymbol{K}_j\big)} \tag{31}$$

where the scaling factor $\sqrt{d}$ has been omitted for the sake of readability. The authors noted that $\sum_{j=1}^{n}\phi\big(\boldsymbol{K}_j\big)\boldsymbol{V}_j^\top$ and $\sum_{j=1}^{n}\phi\big(\boldsymbol{K}_j\big)$ must only be computed a single time, therefore reducing the complexity from quadratic to linear both in terms of memory and computation. The vectorized formulation of the numerator makes it simpler to see:

$$\underbrace{\phi(\boldsymbol{Q})}_{n\times p}\Big(\underbrace{\phi(\boldsymbol{K})^\top}_{p\times n}\underbrace{\boldsymbol{V}}_{n\times d}\Big) \tag{32}$$

where the mapping $\phi(\cdot) : \mathbb{R}^d \to \mathbb{R}^p$ is applied position-wise. Unfortunately, the feature map of the exponential kernel is infinite dimensional. Hence, any finite kernel is an approximation of the attention matrix and may be interpreted as a low-rank factorization. However, they are presented separately here due to their conceptual difference. Katharopoulos et al. [56] approximated the

attention matrix in the Linear Transformer with the feature map $\phi(x) = \text{elu}(x) + 1$, where the function $\text{elu}(\cdot)$ denotes the exponential linear unit given by:

$$\text{elu}(x) = \begin{cases} \alpha(e^x - 1), & x < 0 \\ x, & x \geq 0 \end{cases} \tag{33}$$

where $\alpha$ is an hyperparameter. The Linear Transformer performed on par with the vanilla Transformer on autoregressive image generation, but poorly on automatic speech recognition.

Choromanski et al. [18] later demonstrated that the exponential is equivalent to a kernel with a randomized mapping:

$$\exp(x^\top y) = \mathbb{E}_{w \sim \mathcal{N}(0, I_d)} \left[ \exp\left( w^\top x \frac{\|x\|^2}{2} \right) \exp\left( w^\top y \frac{\|y\|^2}{2} \right) \right] \tag{34}$$

Consequently, the authors introduced the Performer, a linear complexity model that approximates the attention by means of a kernel with the following feature mapping:

$$\phi(x) = \frac{\exp(-\|x\|^2/2)}{\sqrt{2p}} \left[ \exp\left(w_1^\top x\right); \ldots; \exp\left(w_p^\top x\right); \exp\left(-w_1^\top x\right); \ldots; \exp\left(-w_p^\top x\right) \right] \tag{35}$$

where $w_i \sim \mathcal{N}(0, I_d)$. To further reduce the variance of the estimator, $w_i$ are constrained to be exactly orthogonal, which is achieved with the Gram-Schmidt process. The hyperparameter $p$ corresponds to the number of random features and controls the quality of the approximation.

**Clustering and Locality-Sensitive Hashing** [122]: As previously explained, clustering can uncover sparse patterns by grouping queries and keys and only computing the attention between positions within the same cluster. Alternatively, Vyas et al. [122] proposed to factorize the attention with clustering by grouping queries into a fixed number of non-overlapping clusters and by computing the attention between the cluster's centroids and the keys. Consequently, the attention score is only computed once per group of similar queries and broadcasted to all, resulting in linear complexity. Since queries may be clustered differently across attention heads and since the attention sub-layer includes a residual connection, two queries in the same cluster can have different output representations. The authors proved that the approximation error for a given query is bounded by its distance to its centroid multiplied by the spectral norm of the keys matrix. As such, the K-Means algorithm can be used for minimizing the approximation error. However, K-Means in the original space would be slow to compute as Lloyd algorithm has a complexity of $O(ncdl)$, where $c$ is the number of clusters and $l$ is the number of Lloyd iterations. Instead, the authors first used a locality-sensitive hashing scheme on the queries before applying K-Means with the Hamming distance, which reduces the complexity to $O(ncl + cbl + ndb)$, where $b$ is the number of bits used for hashing.

To further improve the approximation, Vyas et al. [122] proposed the *improved cluster attention* that separately consider the $k$ keys with the highest attention for each cluster. Intuitively, keys with high approximated attention may have low attention for some queries, resulting in a large approximation error. As a solution, the dot product between these top-$k$ keys and all queries belonging to the corresponding cluster is computed. Then, the attention is rescaled by the total probability mass assigned to these top-$k$ keys.

Compared to the Reformer, Vyas et al. [122] method is significantly faster (43% lower epoch time) while being significantly more accurate (35% lower phone error rate) for speech recognition on the Wall Street Journal.

### 4.3 Architectural Change

Finally, the Transformer's complexity may also be reduced by modifying the model's architecture and preserving the original attention mechanism. Let us investigate (i) the Transformer-XL and the Compressive Transformer that rely on memory, and (ii) then the Funnel-Transformer that iteratively compresses sequences.

**Memory** [22, 93]: The block-wise approach splits the input sequence into small non-overlapping subsequences called windows, blocks, or chunks, which are processed independently; therefore, the maximum dependency length is equal to that of the subsequence. To leverage information from previous windows, Dai et al. [22] introduced the Transformer-XL, which relies on segment-based recurrence between windows. This recurrence mechanism is implemented by storing the representations of the previous window in a first-in first-out memory (FIFO). Then, the attention mechanism can attend to the representations located in this memory, but the gradients are not computed for the attention on these elements. Although this model achieves a RECL four times greater than the vanilla Transformer with the same parameter budget, it cannot capture dependencies outside the FIFO memory range. Furthermore, this model is only compatible with autoregressive tasks. This technique is analogous to truncated backpropagation through time (BPTT), except that a sequence of hidden states is considered instead of the previous one. Figure 18 illustrates the segment-based recurrence of the Transformer-XL.

In order to further increase the range of dependencies considered by the Transformer-XL, Rae et al. [93] proposed the Compressive Transformer, which adds a compressed memory to the original FIFO memory. Representations of past windows are first stored in the standard FIFO memory, like the Transformer-XL. Then, when this memory is full, the oldest representations are compressed with a user-defined function and stored in the compressed FIFO memory instead of being discarded. The number of elements considered in the original FIFO memory to generate the compressed memory depends on the chosen function. The authors propose using max/mean pooling, 1D convolution, dilated convolutions, or the most attended representations by the attention. They also proposed to learn the compression function with an auxiliary auto-encoding loss and a variant called attention-reconstruction loss, which typically reconstructs the original memory from the compressed ones. They show a clear advantage over the Transformer-XL on NLP tasks and comparable results on speech modelling.

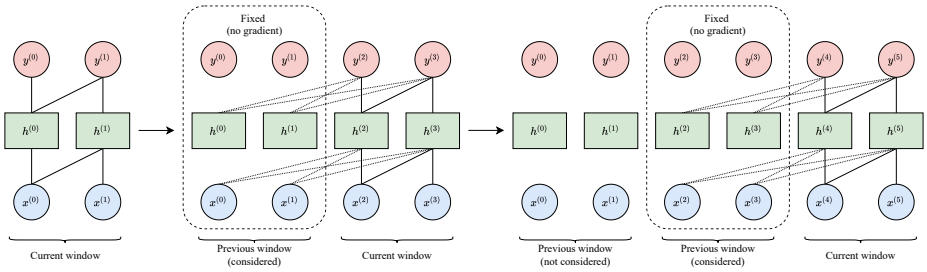


Fig. 18. Segment-based recurrence, which is similar to truncated BPTT. The window size is equal to two, and only the previous window is considered. For the sake of clarity, parameters from and to states that do not contribute are omitted.

**Sequence Compression** [21]: Many tasks such as image classification and sentiment analysis only require producing a single output for the whole sequence. Dai et al. [21] argued that the full-length sequence of hidden states may contain significant redundancy and that the model may

not have to preserve token-level information. Consequently, they proposed the Funnel-Transformer, whose encoder reduces the computational cost by gradually reducing the length of the hidden states sequence with pooling. Note that instead of directly feeding the pooled sequence into the attention layer, it is only used to construct the query matrix, while the unpooled sequence is used to construct the key and value matrices. Additionally, the authors proposed to recover the original sequence length by up-sampling the compressed sequence of hidden states to address the common pre-training objectives, such as MLM, that require separate representation for each token. Although the Funnel-Transformer effectively reduces the computational and memory cost of the encoder, the complexity remains quadratic, and the best performances are achieved on tasks that only require sequence-level representation.

## 5 SHORTCOMINGS

This section discusses the lack of understanding of the self-attention inner workings and the limitation of the Transformer evaluation methodology, including the lack of standard benchmarks for long-range dependencies.

Self-attention is a relatively new mechanism that has been quickly and widely adopted due to its remarkable empirical success. Nonetheless, the self-attention inner workings are not yet fully understood, and many questions remain unanswered, including why it works, what it learns, and whether it is interpretable. Answering those questions is crucial to designing faster and lighter Transformers that are competitive with the original model. As of this paper's writing, the deep learning community actively investigates self-attention and have proposed preliminary answers to the aforementioned questions. For instance, evidence supporting both the self-attention interpretability [101, 131] and non-interpretability [53] have been published. Tay et al. [113] empirically evaluated the dot product impact on natural language processing tasks and concluded that query-keys interaction is "*useful but not that important*". Kitaev et al. [59] investigated the impact of sharing queries and keys, and concluded that "*it turns out that sharing QK does not affect the performance of Transformer*".

Despite our current limited understanding of the self-attention mechanism, a wide range of faster and lighter Transformers have been introduced in a short amount of time, each claiming comparable or superior performance to the vanilla Transformer. Since there is no consensus on how to evaluate the proposed approaches [115], researchers often have to evaluate their method on a small range of tasks. However, different tasks may require different assumptions, which means that one method may work well on a specific task but poorly on others. For instance, Tay et al. [113] showed that a simple Synthesizer is highly competitive with the vanilla Transformer across a range of natural language processing tasks, including machine translation, language modelling, and text generation. However, Tay et al. [115] later showed that the vanilla Transformer outperforms the Synthesizer on the more difficult Long-Range Arena benchmark. Long-Range Arena [115] is a suite of five general and challenging tasks designed to evaluate how well Transformers capture long-term dependencies from different modalities such as text, natural and synthetic images, and mathematical expressions. Table 3 compiles the Long-Range Arena results of the models discussed in the survey. For a complete description of the objectives and datasets, we refer the reader to the original paper.

Furthermore, due to Transformers large training cost, researchers often evaluate their approach against a limited number of models on the tasks of interest. For instance, [59] only evaluated the Reformer against three distinct vanilla Transformers [85, 121] on three tasks. Standardized suites of benchmarks such as GLUE and the recent Long-Range Arena allow researchers and practitioners to evaluate only their method and compare it against a public leaderboard. Consequently, we highly recommend that researchers consider such benchmarks.

Although standardized benchmarks such as Long-Range Arena would help compare the models, the results should be taken with caution since the performance depends on the model size and hyperparameters, the speed depends on the implementation and hardware, and the memory footprint depends on the implementation and general methods used. For instance, the Switch Transformer uses a mixture of experts, mixed-precision, expert dropout, knowledge distillation, and a careful initialization. Therefore, it is difficult to isolate the benefit of a single modification.

Finally, the complexity is not always representative of the practical efficiency. For instance, the Reformer achieves an asymptotic complexity of $O(n \log n)$ but is significantly slower than the vanilla Transformer on small sequences, as shown in Table 3. This slow down is due to large constants hidden in the complexity. Even when there are no hidden constants, there is a distinction between theoretical complexity and what is achievable in practice. For instance, sparse matrix multiplication may reduce the complexity from quadratic to linear in theory. However, it is well known that GPUs and TPUs are not designed to perform such operations efficiently [11] and, in practice, sparse matrix multiplication is often slower than dense ones. We encourage researchers to explicitly report the complexity as well as the number of floating operations (FLOPs), the wall-clock time with the hardware, and the memory footprint of their method.

Table 3. Long-Range Arena benchmark [115]. Results have been compiled from the original paper. Benchmarks are run on 4x4 TPU V3 chips, and the memory is reported per device.

| Models | Average score (%) | Steps per second | | Peak memory (GB) | |
|---|---|---|---|---|---|
| | | 1K | 4K | 1K | 4K |
| Transformer [121] | 54.39 | 8.1 | 1.4 | 0.85 | 9.48 |
| Sparse Transformer[9] [16] | 51.24 | | | | |
| Longformer[9] [5] | 53.46 | | | | |
| BigBird [139] | **55.01** | 7.4 | 1.5 | 0.77 | 2.88 |
| Sinkhorn Transformer [114] | 51.39 | 9.1 | 5.3 | 0.47 | 1.48 |
| Reformer [59] | 50.67 | 4.4 | 1.1 | 0.48 | 2.28 |
| Linformer [126] | 51.36 | 9.3 | 7.7 | **0.37** | **0.99** |
| Synthesizer [113] | 51.39 | 8.7 | 1.9 | 0.65 | 6.99 |
| Linear Transformer [56] | 50.55 | 9.1 | 7.8 | **0.37** | 1.03 |
| Performer [18] | 51.41 | **9.5** | **8.0** | **0.37** | 1.06 |

## 6 BROADER IMPACT OF EFFICIENT TRANSFORMER

This section extends the three motivations and potential impacts of lighter and faster Transformers briefly discussed in Section 2.4.

First and foremost, computational resources are not only finite but also expensive. Consequently, there are severe inequalities between research groups and between companies. Indeed, many researchers do not have access to GPU or TPU farms, and most companies cannot afford to spend thousands or millions of dollars on dedicated hardware, especially if deep learning is not their primary focus. At this time, the resources disparities have increased dramatically to a point where only a few parties can afford to train massive state-of-the-art models. A prime example of this cleavage is the Transformer. Indeed, the largest Transformers are so expensive to train, even for large companies such as Microsoft, that they are only trained once. For instance, Brown et al. [10] noticed an issue in their pre-processing after training GPT-3. As the author explained, they could not train their model again due to the massive cost and therefore published their results with a

[9]The Sparse Transformer and Longformer depends on CUDA kernels that are difficult to implement on TPUs. Therefore, Tay et al. [115] used *equivalent* implementations to emulate their performance and did not report their efficiency.

known issue. Resources inequalities also hinder creativity as researchers with promising ideas may not be able to implement them, thus reinforcing the vicious "rich get richer" circle, where well-funded groups and companies that have access to more resources are more likely to achieve state-of-the-art results and receive more fundings [108].

Additionally, lower-complexity Transformers enable novel applications as extremely long sequences cannot be processed in a reasonable amount of time by the quadratic complexity vanilla Transformer. For instance, Choromanski et al. [18] observed the Performer's potential impact on biology, and Zaheer et al. [139] evaluated BigBird on genomics tasks that take fragments of DNA as input. Huang et al. [46] were able to generate minute-long musical compositions with a Transformer that leverage the block-wise approach and an efficient computation of the relative attention. Note that contrary to the attention introduced by [121], the relative attention [102] explicitly models the input positions. The range of applications will surely expand as researchers design ever-lighter and -faster Transformers.

Finally, recent research made it clear that we must cut carbon dioxide (CO2) emissions in half over the next decade to limit global warming. The large-scale infrastructures used by the deep learning community consume a considerable amount of electricity, which is mainly produced by non-renewable sources such as coal or gas [49]. Strubell et al. [108] estimated that training a Transformer with neural architecture search generates up to 284,000 kg of CO2. For reference, the average American emits 16,400 kg of CO2 per year, and the average car emits about 57,200 kg during its lifetime[10] (fuel included). The authors estimated that training a single instance of BERT [24] on GPU produces about the same amount of CO2 as a trans-American flight. Although lighter and faster models require fewer resources and therefore produce less carbon dioxide, they are also more accessible, so we would expect more models to be trained. Overall, it is difficult to know whether lighter and faster Transformers will positively impact the environment. Nonetheless, researchers and practitioners ought to have in mind the significant environmental impact of their experiments, which can be estimated with the Machine Learning Emissions Calculator[11] developed by Luccioni et al. [75].

## 7 FUTURE RESEARCH DIRECTIONS

In our opinion, the current research directions follow one of two purposes: (i) efficiency and affordability or (ii) generalization performance. Since this survey addresses approaches to yield faster and lighter Transformers, let us start with the efficiency and affordability objective.

### 7.1 Efficiency and Affordability

To the best of our knowledge, researchers and practitioners have not yet identified a specialized approach that improves the Transformer's efficiency for every task, dataset, and hardware, as explained in Section 5. In our opinion, one of the most promising avenues is to learn adaptively sparse patterns that are structured for the available hardware. Let us justify our claim.

The Softmax function only contains a few large values due to its exponential nature. Therefore, it can be effectively approximated by masking the positions with small weights. In theory, the computation and memory reduction is linearly proportional to the ratio of masked positions. In practice, however, the improvement depends on the hardware. As of this survey's writing, NVIDIA is the first and only manufacturer to offer an architecture that natively supports sparse operations, resulting in a virtually perfect speed-up. One may reasonably expect other manufacturers to follow this direction due to the prevalence of sparse operations in deep learning. Therefore, the sparse

[10] A product lifetime or lifecycle typically includes material production, manufacturing, usage, and end-of-life disposal.
[11] https://mlco2.github.io/impact/

patterns should be structured such that the hardware natively supports them. Handcrafting features or patterns based on prior knowledge is known to be suboptimal. Instead, the model should learn the patterns from the data for the task at hand. Additionally, individual samples are likely to require different attention patterns, and hence, the patterns should be adaptative (content-based). Finally, we believe it is beneficial to include global tokens since they allow any position to attend to any other position in two layers, thus preserving the attention's expressiveness.

### 7.2 Generalization Performance

A second research venue consists in improving the network generalization performance. Since the deep learning renaissance associated with greedy layer-wise unsupervised pre-training [36], there has been a clear trend towards scaling up neural networks. As a result, researchers and practitioners have been able to leverage ever-larger datasets and ultimately improve the network's performance. In this setting, scaling is performed typically by increasing the number of layers, the number of attention heads, the input embedding dimension, and the feedforward network width.

Amongst others, Radford et al. [92] introduced a large Transformer called GPT-2 and evaluated various model sizes on language modelling tasks in a zero-shot setting. The authors reported that the performance significantly increased with the model size ranging from 117M to 1.5B parameters. Recently, Brown et al. [10] introduced GPT-3 based on the GPT-2 architecture and considered an even wider span of model sizes, ranging from 125M to 175B parameters. The authors reported that the model performance smoothly increased with the model size in most cases and suggested that this trend should extend to even larger models. Furthermore, Devlin et al. [24] investigated the effect of BERT size on the GLUE benchmark and concluded that "*larger models lead to a strict accuracy improvement across all four datasets, even for MRPC which only has 3,600 labeled training examples, and is substantially different from the pre-training tasks*".

These observations suggest that researchers and practitioners must scale their model to pursue the generalization performance objective. Inherently, scaling is resource-expensive and goes against the affordability sought in this survey. Nonetheless, there are research directions to improve the generalization capability of deep learning models that are orthogonal to scaling and thus compatible with efficiency. A promising avenue is structural inductive biases. A recent structural inductive bias inspired by independent mechanisms in the causality literature consists of designing an architecture that learns sparsely interacting modules, each one of them specialized in a different mechanism [37]. Ideally, individual modules should be robust to changes in the aspects of the world that are unrelated to this module, such as in the case of distributional shift. Lamb et al. [61] applied this idea to Transformers by introducing the Transformers with Independent Mechanisms (TIM). The authors observed that TIM layers could be combined with the mixture of experts approach, allowing the switching to be specific to distinct aspects of the data.

Combining universally effective and efficient approaches such as the aforementioned sparse patterns with conditional computing and the independent mechanisms prior appears to be promising to tackle complex tasks without relying on large-scale resources.

## 8 CONCLUSION

Transformers have quickly become the de facto model for processing sequences, notably achieving state-of-the-art in most natural language processing tasks at the cost of quadratic complexity. As a result, researchers have leveraged numerous techniques to mitigate this memory and computational burden. This survey investigated popular general methods to make neural networks lighter and faster and discussed their strengths and limitations. Notably, we advised researchers and practitioners to use mixed-precision and gradient checkpointing due to their simplicity and overall