Computationally Efficient Architectures for NLP: The Transformer and its Competitors

Kilian Brickl, Roman Alyaev

Abstract

Transformers have revolutionized natural language processing (NLP) and other domains due to their scalability and exceptional performance on various tasks. However, their high computational cost and memory requirements often limit their applicability, particularly in real-time and resource-constrained environments. This paper provides a short introduction to older architectures for NLP tasks and then gives an insight into the mechanisms of the Transformers and a few of its relevant extensions. Subsequently, we show alternative architectures that are potential candidates to surpass the performance of Transformers in the future, like Attention-Free Transformers, RWKV, and State Space Models. Finally, we compare the architectures to highlight the trade-offs between runtime behavior and model expressiveness, offering insights into selecting architectures based on application-specific constraints. This work aims to provide an overview of the current developments in Transformer architectures and give a sense of what architectures could be used for NLP tasks beyond Transformers.

Index Terms

NLP, Transformer, Attention, Attention-Free Transformer, RWKV, State Space Models.

I. Introduction

NATURAL language processing (NLP) is a field at the intersection of computer science, artificial intelligence, and linguistics, concerned with the interactions between computers and human languages. The goal of NLP is to enable computers to understand, interpret, and generate human language in a way that is meaningful and useful.

Early NLP systems relied heavily on hand-crafted rules and symbolic methods [1], [2]. These systems, such as Weizenbaum's ELIZA [1] and Winograd's SHRDLU [2], utilized predefined patterns and logical structures to process language inputs. While innovative at the time, these approaches faced scalability issues due to the complexity of language and the extensive effort required to encode linguistic knowledge manually [3], [4]. With the advent of machine learning, statistical approaches began to dominate, leading to more flexible and robust models. The introduction of deep learning further revolutionized NLP by enabling models to learn hierarchical representations of data. Currently, these models are widely popular due to their broad range of applications. In this paper, we would like to give an introduction to the currently most used architectures and what could be potential successors to them in the future.

II. Importance of Word Embeddings

Processing natural language requires translating human words into representations that machines can understand. These representations must also capture the meanings and relationships between words. To achieve this, models use vector representations, known as word embeddings, which encode semantic and syntactic information. Word embeddings map words or phrases from a vocabulary into numerical vectors that are learned from data and capture the contextual relationships in which words appear.

In this context, a *token* refers to the fundamental unit of text processed by NLP models. Depending on the tokenization strategy, tokens may represent entire words, subwords, characters, or other linguistic elements. For instance, traditional models like Word2Vec [5] and GloVe [6] treat words as tokens, while modern subword-based models, such as BERT, decompose words into smaller units or pieces, enabling more flexibility in handling rare or unseen words.

III. Classical architectures for sequence modeling tasks

Predominant architectures for sequence modeling tasks in NLP, prior to architectures like Transformers, were Recurrent Neural Networks (RNNs) and Convolutional Neural Networks (CNNs). As they are still used today in various extensions, we dedicate the following section to them.

A. Recurrent Neural Networks

RNNs are designed to handle sequential data by maintaining a hidden state that captures information about previous elements in the sequence, i.e. the history. For every new input, the knowledge about the sequence history is updated by adapting the hidden state with the new information (Figure 1).

The RNN is fundamentally a natural model for NLP tasks as the next token is always generated based on the known history autoregressively. As soon as an output is generated, the hidden state is updated to capture the new information. But this also

means that all information about the context is captured in one single hidden state vector. This obviously is a limitation of the expressiveness of the model and also causes the well-known problem of vanishing and exploding gradients. Recurrent models usually have difficulties in learning long-range dependencies as the gradients, which are needed for training, become too small or too large during backpropagation. Another problem is the inability to train RNNs in parallel as they work in an autoregressive fashion by predicting one token at a time. This makes it impossible to make use of modern computer architecture and therefore training takes comparably long [7].

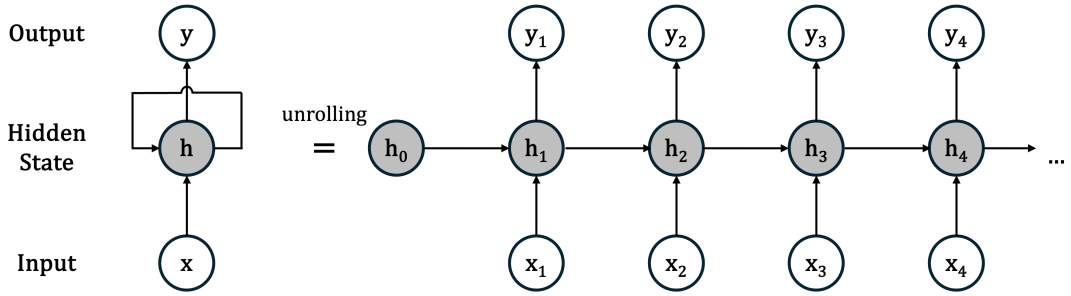

Fig. 1. Recurrent Neural Network (RNN) architecture unrolled over time.

B. Convolutional Neural Networks in NLP

Originally successful in computer vision, CNNs (Figure 2) were adapted for NLP tasks such as sentence classification [8]. The idea was to use convolution over word embeddings to capture local features.

The advantages of CNNs are the ability to compute convolutions in *parallel* over different parts of the input and to capture *local dependencies*. However, there are also limitations, for example the fixed receptive field which is the reason why CNNs require the stacking of many layers or using large kernels to capture long-range dependencies. Another caveat is the positional information as convolutions alone do not inherently capture the position of words in a sequence.

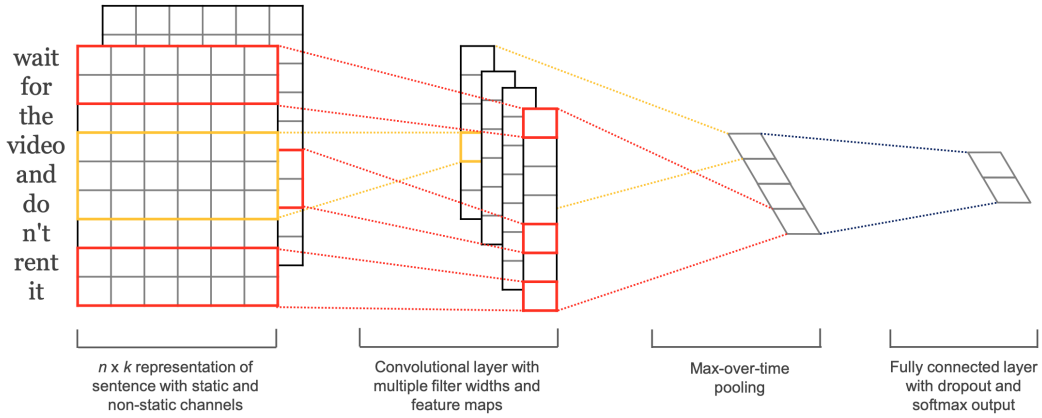

Fig. 2. Convolutional Neural Network (CNN) Architecture with two channels for an example sentence from [8]

Max-over-time pooling operation, from Figure 2, captures the most significant feature across a dimension, making it a crucial operation in CNN-based NLP tasks for extracting salient features. This technique works by sliding a window over a sequence of embeddings and retaining only the maximum value from each window. This operation ensures that only the most salient features from the input are retained, effectively reducing the dimensionality while preserving the most informative characteristics.

IV. Transformers

NLP has witnessed significant advancements with the development of deep learning models like RNNs and CNNs capable of understanding and generating human language. However, they have limitations in handling long-range dependencies and computational efficiency. A huge breakthrough was the invention of the attention mechanism and the transformer architecture. The attention mechanism allows models to focus on specific parts of the input when generating output.

Bahdanau et al. [9] introduced attention to encoder-decoder models, improving performance in machine translation. This mechanism enables processing inputs of various lengths and improves long-range dependency modeling by directly connecting

output tokens with relevant input tokens. The Transformer introduced by Vaswani et al. [10] leverages the self-attention mechanism and parallel processing capabilities, raising this architecture to the current state-of-the-art model for nearly all applications in NLP.

A. Transformer Architecture

The transformer architecture eliminates the need for recurrence and convolution, relying entirely on attention mechanisms to draw global dependencies between input and output. The idea of the attention mechanism is that the model is provided with the ability to attend to different positions in the input sequence, capturing relationships and dependencies between words. *Scaled Dot-Product Attention* computes

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{Q \cdot K^T}{\sqrt{d_k}}\right) \cdot V. \tag{1}$$

For this computation, three matrices are created by multiplying the input sequence $X \in \mathbb{R}^{n \times d_{\text{model}}}$ with different learnable weight matrices. Here, n represents the length of the input sequence and d_{model} the dimension of the embedding space.

$$\begin{aligned} \text{Key:} \quad & K = X \cdot W^K \quad , W^K \in \mathbb{R}^{d_{\text{model}} \times d} \\ \text{Query:} \quad & Q = X \cdot W^Q \quad , W^Q \in \mathbb{R}^{d_{\text{model}} \times d} \\ \text{Value:} \quad & V = X \cdot W^V \quad , W^V \in \mathbb{R}^{d_{\text{model}} \times d} \end{aligned} \tag{2}$$

One interpretation that might help to better understand the math is the following: The query represents the element e we are trying to seek information about. The key is used to determine which of the other elements are relevant to get information about e. The value contains the actual information from the other elements we want to get to further understand the true meaning of e.

Transformer consists of an encoder and a decoder (Figure 3 left), both composed of multiple stacked layers.

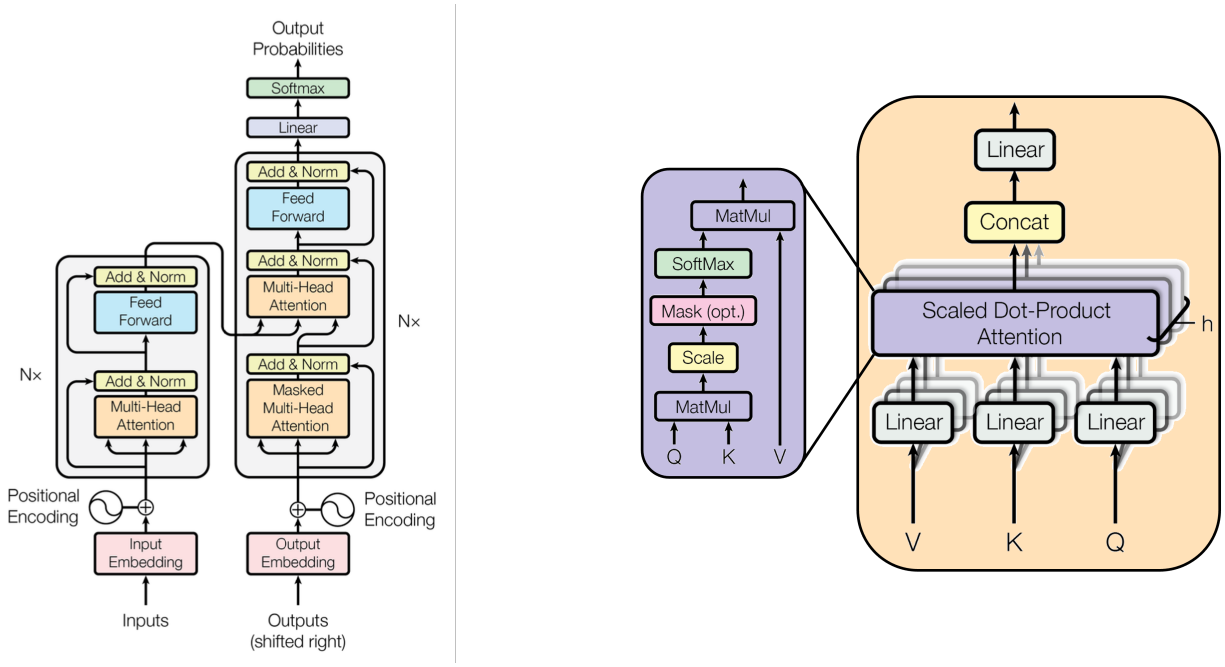

Fig. 3. (left) A full encoder-decoder transformer architecture. (right) A multi-head attention block with h scaled dot-product attention blocks. [10]

Instead of performing only one scaled dot-product attention, Vaswani et. al. [10] introduce the idea of multi-head attention, which is a first part of each encoder layer. Therefore, the three matrices (Equation 2) are split up into different channels of size $d_k = d_{\text{model}}/h$ with h being the number of parallel scaled dot-product attention blocks as shown in Figure 3 on the right side. The idea behind this is that different subspaces of the total embedding space represent different aspects of the actual token, hence splitting up the learning process allows to learn more complex structures. Each single head gets its own, separate chunk of the matrices $K_i, Q_i, V_i \in \mathbb{R}^{d_k}$ with $i = 1, ..., h$. Afterwards, the output of all h attention heads are concatenated so the output of one multi-head attention block has again size $n \times d_{\text{model}}$.

At the end of one encoder layer, a *Position-Wise Feed-Forward Network* is applied to each position individually to further enhance information exchange. Both sub-layers employ residual connections and layer normalization to facilitate training.

The decoder is also composed of a stack of N identical layers as the encoder. However, the decoder has three sub-layers, incorporating a masked multi-head attention block in addition the the normal multi-head attention block and the feed-forward network. The *Masked Multi-Head Attention* extends the normal multi-head attention by a masking mechanism. This ensures that the encoder cannot "see" any future tokens, ensuring the model avoids predicting output using future knowledge. This is done by setting all values in the matrix $Q \cdot K^T$, which correspond to future tokens, to $-\infty$ (see the pink block in the middle of Figure 3). This will ensure that the SoftMax activation function sets these values to zero which means they are not considered any further.

The decoder's output mechanism comprises a linear layer followed by a SoftMax activation function. This produces a probability distribution over the entire vocabulary, allowing the model to select the most probable next token. The encoder and decoder differ in their use of attention. While the encoder's self-attention focuses on learning relationships within the input sequence, the decoder uses cross-attention to align generated tokens with encoder outputs. This interplay ensures that generated text remains coherent and contextually relevant.

B. Positional Encoding

To retain the order of the sequence, positional encodings are added to the input embeddings. These encodings use sine and cosine functions of different frequencies:

$$\begin{cases} \text{PE}(\text{pos}, 2i) = \sin\left(\frac{\text{pos}}{10000^{2i/d_{model}}}\right); \\ \text{PE}(\text{pos}, 2i+1) = \cos\left(\frac{\text{pos}}{10000^{2i/d_{model}}}\right); \end{cases} \tag{3}$$

where pos is the position and i is the dimension. That is, each dimension of the positional encoding corresponds to a sinusoid. The wavelengths form a geometric progression from 2π to $10000 \cdot 2\pi$. The positional encoding is necessary as the meaning of words also depends on their position in a sentence.

C. Computational Efficiency of Standard Transformers

The transformer architecture offers significant computational benefits over traditional models. Firstly, transformers process all positions of the input sequence simultaneously, enabling efficient utilization of parallel computing resources like GPUs and TPUs. This contrasts with RNNs, which process sequences sequentially and cannot leverage parallelism effectively.

Additionally, the self-attention mechanism allows the model to capture dependencies between any two positions in the sequence, regardless of their distance. This capability addresses the limitations of RNNs and CNNs in modeling long-range dependencies.

When it comes to computational complexity, the self-attention mechanism has complexity $O(n^2 \cdot d)$ because of the matrix-matrix multiplication $Q \cdot K^T$, where n is the sequence length and d is the embedding dimension. While this can be computationally intense for long sequences, the absence of sequential operations enables efficient computation on modern hardware. Memory efficiency can also pose challenges for long sequences since it is $O(n^2)$.

On top of that, there are a couple more challenges for transformers. Transformers require large amounts of data to train effectively. Exploring methods for improving data efficiency, such as transfer learning and unsupervised pre-training, is an ongoing area of research. And lastly, understanding how transformers make decisions is challenging due to their complex attention patterns. Enhancing interpretability will help in debugging models and ensuring fairness.

V. Extensions of the Transformer Architecture

The original Transformer architecture has significantly advanced natural language processing by enabling efficient parallel computation and effective modeling of long-range dependencies. However, its quadratic time and memory complexity with respect to the sequence length n poses challenges when dealing with very long sequences. To address these limitations, various extensions have been proposed. This section focuses on two substantial approaches: *Reformer* and *Performer*. These models aim to reduce the computational complexity and memory requirements of the Transformer, making it more scalable and efficient for long-sequence processing.

A. Reformer

Reformer [11] introduces two key innovations to tackle the Transformer's quadratic complexity: Locality-Sensitive Hashing (LSH) Attention and Reversible Residual Layers. These modifications reduce the time and memory complexity from $\mathcal{O}(n^2)$ to $\mathcal{O}(n \log n)$, enabling the model to handle much longer sequences efficiently.

The core idea of *LSH attention* is to approximate the dot-product attention mechanism using hashing, which groups similar queries and keys together, reducing the number of pairwise comparisons. In the vanilla Transformer, the scaled dot-product attention is computed as (1). The computation of $QK^\top$ requires $\mathcal{O}(n^2)$ operations. Reformer replaces the full attention computation with an approximate method using LSH. The steps are as follows:

- Hashing Queries and Keys: Both queries and keys are hashed into h buckets using random projections. The hash function $h : \mathbb{R}^{d_k} \to \{1, \ldots, B\}$ maps similar vectors to the same bucket with high probability.
- Within-Bucket Attention: Attention is computed only among queries and keys that fall into the same bucket.
- Reducing Complexity: Since each bucket contains approximately n/B elements, the overall complexity reduces to $\mathcal{O}(n \log n)$. The complexity does not explicitly depend on B because B determines the trade-off between the number of buckets and bucket size, influencing hashing efficiency rather than the computational formula.

Let $\mathcal{B}_b$ denote the set of indices in bucket b. The attention for query q_i is computed as:

$$\text{Attention}(q_i, K_{\mathcal{B}_b}, V_{\mathcal{B}_b}) = \sum_{j \in \mathcal{B}_b} \alpha_{ij} v_j, \tag{4}$$

where α_{ij} is the attention weight between q_i and k_j within the same bucket, computed as:

$$\alpha_{ij} = \frac{\exp\left(\frac{q_i^\top k_j}{\sqrt{d_k}}\right)}{\sum_{l \in \mathcal{B}_b} \exp\left(\frac{q_i^\top k_l}{\sqrt{d_k}}\right)}. \tag{5}$$

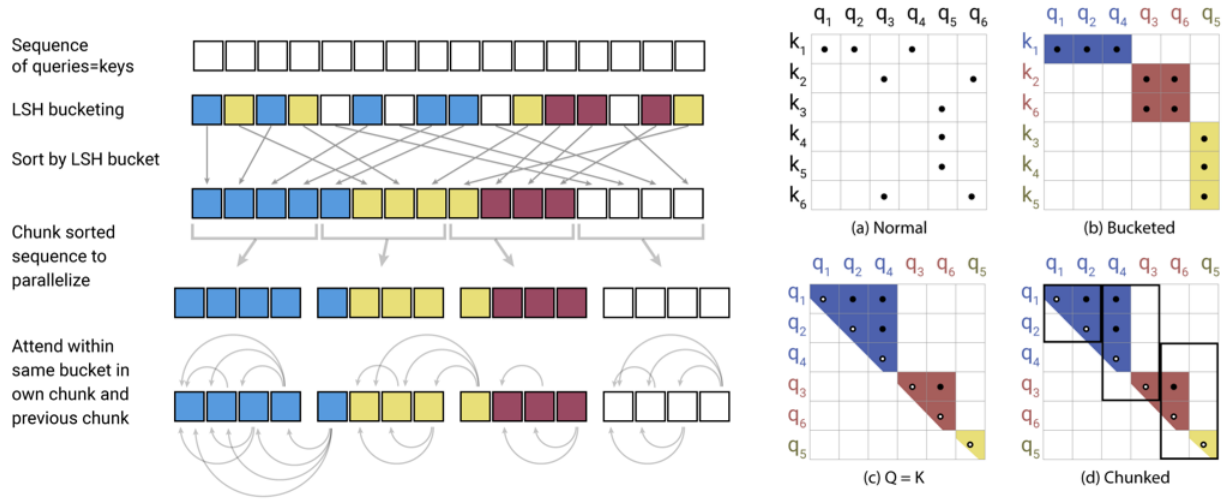

Fig. 4. LSH Attention mechanism in Reformer: Queries and keys are hashed into buckets, and attention is computed within each bucket. This reduces the complexity from $\mathcal{O}(n^2)$ to $\mathcal{O}(n \log n)$ [11].

Figure 4 visualizes the Reformer's LSH attention mechanism. The input sequence is divided into segments, and tokens are hashed into buckets based on similarity. Within each bucket, tokens are sorted by hash values to enhance data locality. Attention is then computed only within these buckets, reducing the complexity from quadratic to logarithmic. This figure highlights how LSH enables efficient attention by limiting computations to localized groups, making it scalable for long sequences.

In addition to bucketing, Reformer employs *reversible residual layers* (Figure 5) to minimize memory usage during training. Unlike standard residual connections, which require storing intermediate activations for backpropagation, reversible layers enable reconstruction of inputs from outputs. This approach significantly reduces memory overhead while maintaining the expressive power of residual connections. In the vanilla Transformer, each layer's output is computed as:

$$\mathbf{y} = \mathbf{x} + \text{Layer}(\mathbf{x}), \tag{6}$$

requiring storage of activations $\mathbf{x}$ for backpropagation, leading to $\mathcal{O}(nL)$ memory usage, where L is the number of layers. In reversible layers, the input can be reconstructed from the output, eliminating the need to store activations:

$$\mathbf{y}_1 = \mathbf{x}_1 + \text{F}(\mathbf{x}_2), \tag{7}$$

$$\mathbf{y}_2 = \mathbf{x}_2 + \text{G}(\mathbf{y}_1). \tag{8}$$

In Equations 7 and 8, x_1 and x_2 represent the input activations split across two parts at each layer. These components are processed separately by the functions F and G, which correspond to the operations applied within the reversible residual layer. This division allows partial computations to be reversed during backpropagation.

During backpropagation, $\mathbf{x}_1$ and $\mathbf{x}_2$ can be recovered from $\mathbf{y}_1$ and $\mathbf{y}_2$, reducing memory usage to $\mathcal{O}(n)$. What's more, it maintains the same expressive power as standard residual layers.

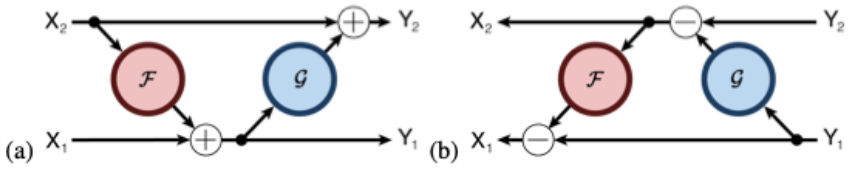

Fig. 5. Reversible residual layers in Reformer: The inputs can be reconstructed from the outputs, reducing memory usage during backpropagation [12].

B. Performer

Performer [13] addresses the quadratic complexity by introducing a linear attention mechanism called *Fast Attention Via Positive Orthogonal Random features (FAVOR+)*. This method approximates the softmax attention using kernel feature maps, achieving linear time and space complexity. The key idea is to express the softmax attention as a kernel function and approximate it using a feature map. The standard attention can be rewritten using a kernel function $A(i,j) = K(q_i, k_j) = \exp\left(q_i^\top k_j\right)$:

$$\text{Attention}(Q,K,V) = D^{-1}\left(\Phi(Q)\left(\Phi(K)^\top V\right)\right), \tag{9}$$

where $\Phi(\cdot)$ is a feature map such that $\Phi(q_i)^\top \Phi(k_j) = K(q_i, k_j)$, and D is a diagonal matrix for normalization.

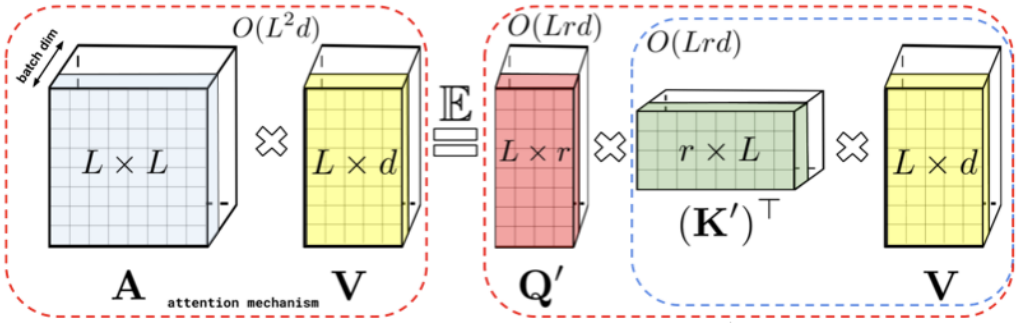

Fig. 6. Performer linear attention mechanism: Approximation of the regular attention mechanism AV (before D^{-1}-renormalization) via (random) feature maps. Dashed-blocks indicate order of computation with corresponding time complexities attached [13].

The above scheme constitutes the FA-part of the FAVOR+ mechanism. The remaining OR+ part answers the following questions:

- How expressive is the attention model defined in $\Phi(q_i)^\top \Phi(k_j) = K(q_i, k_j)$, and in particular, can we use it in principle to approximate regular softmax attention?
- How do we implement it robustly in practice, and in particular, can we choose $r \ll L$ for $L \gg d$ to obtain desired space and time complexity gains?

Performer uses a positive random feature map $\Phi(\cdot)$ to approximate the exponential kernel:

$$\Phi(x) = \exp\left(-\frac{\|x\|^2}{2}\right)\exp\left(\omega^\top x\right), \tag{10}$$

where ω is a random vector drawn from a suitable distribution. Using the approximation, attention is computed as:

$$\text{Attention}(Q,K,V) = \Phi(Q)\left(\Phi(K)^\top V\right), \tag{11}$$

without the need to compute $QK^\top$, reducing complexity to $\mathcal{O}(nd_k^2)$. The normalization is adjusted to account for the approximation:

$$\text{Attention}(Q,K,V) = \frac{\Phi(Q)\left(\Phi(K)^\top V\right)}{\Phi(Q)\left(\Phi(K)^\top \mathbf{1}\right)}, \tag{12}$$

where $\mathbf{1}$ is a vector of ones.

This mechanism enables handling very long sequences because of its linear complexity in time and space and can replace standard attention mechanisms without significant architectural changes.

Figure 7 compares the speed and memory usage of various transformer variants across sequence lengths. It highlights the trade-offs between computational efficiency and expressiveness, showcasing Performer as efficient for longer sequences.

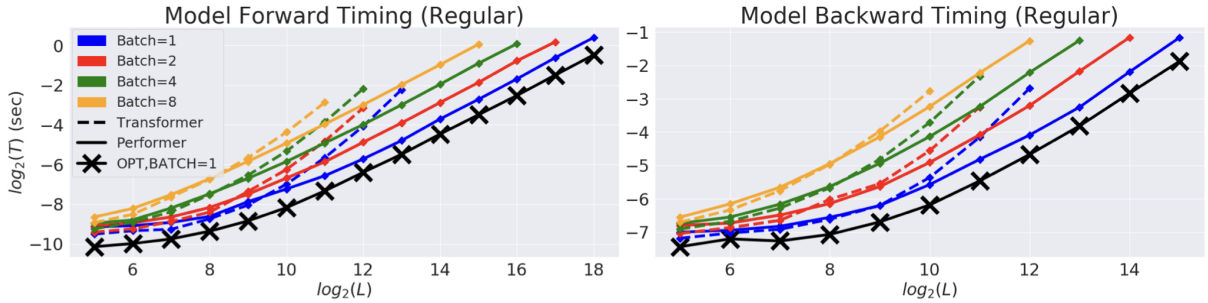

Fig. 7. Comparison of Transformer and Performer in terms of forward and backward pass speed and maximum L allowed. "X" (OPT) denotes the maximum possible speedup achievable, when attention simply returns the V-matrix. Plots shown up to when a model produces an out of memory error on a V100 GPU with 16GB. Vocabulary size used was 256. Best in color [13].

C. Comparison and Discussion

Both models significantly reduce the computational (quadratic) complexity of the attention mechanism, enabling processing of sequences that are orders of magnitude longer than what vanilla Transformer can handle efficiently. Reformer reduces complexity to $\mathcal{O}(n \log n)$ using LSH attention (approximating attention via limiting computations to similar queries and keys), whereas Performer achieves $\mathcal{O}(n)$ complexity via direct kernel approximations. When it comes to memory efficiency, Reformer uses reversible layers to reduce memory consumption. Performer maintains standard Transformer memory usage but benefits from linear attention.

Performer and Reformer could theoretically be combined, leveraging Reformer's LSH-based attention with Performer's kernel approximation. Challenges include managing conflicting optimization strategies and increased architectural complexity.

Reformer and Performer open up possibilities for applying Transformer models to tasks involving very long sequences, such as long document summarization, DNA sequence analysis, and more.

VI. Attention-Free Architectures

Transformers and all its extensions are the go-to architectures in modern NLP applications. However, they have limitations. Especially the quadratic scaling in the sequence length for the attention mechanism drives further research about architectures that are independent of the attention mechanism introduced by Vaswani et. al [10] in 2017. As there are quite a few promising results, we want to present an alternative to the attention mechanism as well as two foundation models that shown promising results in test scenarios and hence could be potential competitors to the currently unassailable transformer architecture.

A. Attention-Free Transformer

First, we want to present an alternative to the multi-head attention deployed in most of the currently used transformer models, the Attention-Free Transformers (AFT) from Zhai et. al. [14]. The AFT model eliminates the quadratic runtime caused by the attention mechanism. To see the transition to the AFT, we repeat the calculation of the attention mechanism below in equation (13). The additional index i represents the i-th head from a multi-head attention (MHA), i.e. $f_i(X_i)$ is the output of the i-th head. The output of one MHA block is the concatenation of the outputs f_i of all heads, with f_i being calculated as

$$f_i(X) = \text{softmax}\left(\frac{Q_i \cdot K_i^T}{\sqrt{d_k}}\right) \cdot V_i \tag{13}$$

with the key, query and value matrix computed individually for each attention head in the MHA:

$$Q_i = XW_i^Q, \quad K_i = XW_i^K, \quad V_i = XW_i^V. \tag{14}$$

In contrast to this, the AFT takes the full key, query and value matrices:

$$Q = XW^Q, \quad K = XW^K, \quad V = XW^V \tag{15}$$

The output of a attention-free transformer block for one position in time is then calculated as:

$$Y_t = \text{sigmoid}(Q_t) \odot \frac{\sum_{j=1}^{T} \exp(K_j + w_{t,j}) \odot V_j}{\sum_{j=1}^{T} \exp(K_j + w_{t,j})} \tag{16}$$

A visualization of this equation is shown in Figure 8. Y_t is the output corresponding to one single input embedding, e.g. a word embedding. To get outputs for a whole sequence, the output of all time steps are concatenated. This is in contrast to MHA where the output of a single head is a certain channel over all embeddings from the input and the individual channels need to be concatenated to get the output.

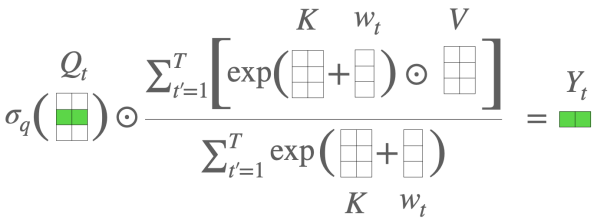

Fig. 8. Visualization of equation 16 with $T = 3, d = 2$ from [14].

The huge advantage of AFT is that there is no dot product between the query and key matrix which is the cause of the $O(n^2)$ scaling in standard transformers. Instead, the *pair-wise position biases* $w \in \mathbb{R}^{\mathbb{T}\times\mathbb{T}}$ are used to achieve global context awareness. In practice, the attention-free transformer block can be used as a one-to-one replacement of the standard multi-head attention block without changing anything else in the architecture. Remarkably, the architecture with the AFT block performs on a level comparable to the full MHA block in test scenarios.

B. Receptance Weighted Key Value (RWKV)

In 2023, Peng et. al. [15] extended the idea of attention-free transformers and combined it with recurrent neural networks which resulted in the RWKV architecture. The idea of using recurrent networks for NLP tasks is widespread. However, as already mentioned in chapter III, nearly all of the existing approaches face the problem of exploding or vanished gradients which are caused by the backpropagation through time [7]. Architectures like the LSTM from Hochreiter and Schmidhuber [16] or GRU from Chung et. al. [17] try to overcome these issues. However, these models still need sequential input. This renders it impossible to utilize modern, highly parallelized hardware making these models less effective than the transformer.

In contrast, the RWKV model combines the sequential processing strengths of RNNs with the parallelism and representational power of transformers. To achieve this, the RWKV substitutes the multi-head attention with so-called time-mixing and channel-mixing blocks. These blocks can be either written in a parallel formulation for fast training or in a recurrent formulation for fast inference, enabling faster computation and reduced memory requirements, particularly for long-sequence tasks. This duality in the formulation is the big strength of the RWKV compared to transformers.

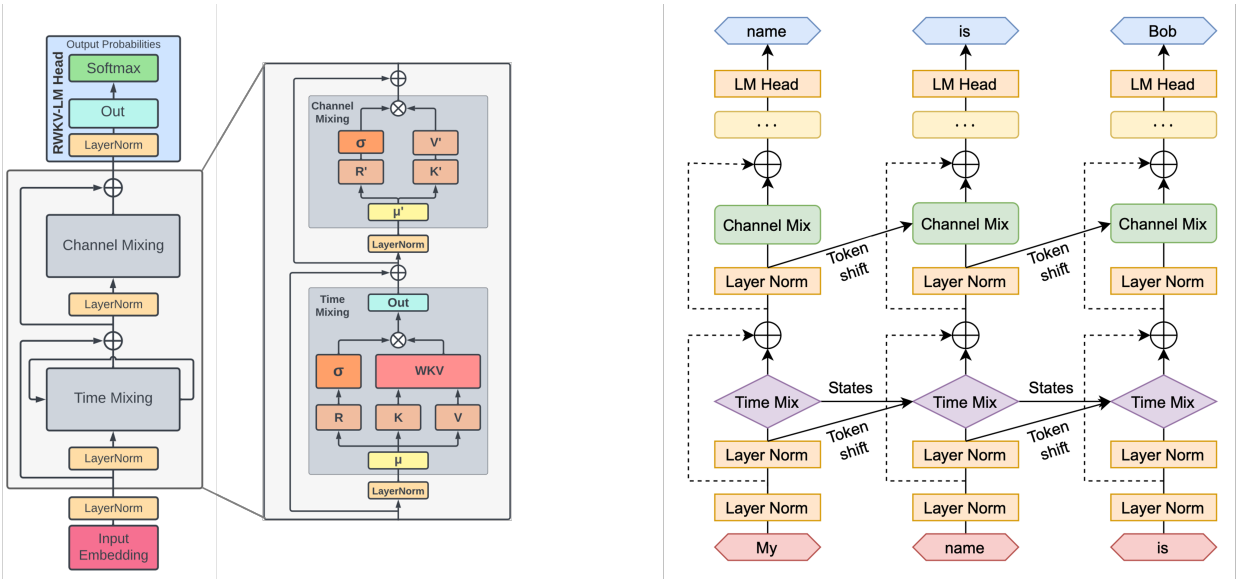

Fig. 9. Complete architecture of a RWKV language model, on the left a RKWV residual block and on the right the unrolling of the residual block over time for an input sequence "My name is" from [15].

Figure 9 shows a complete RWKV architecture with its time mixing and channel mixing blocks. To have access to tokens from past time steps, the RWKV model includes token-shifting. At each time step, the states inside the time-mixing and channel-mixing block are a linear combination of the current and the previous token. The ratios μ with which these two tokens

are added together are learned parameters. Together with the weight matrices W_r, W_k and W_v, the output of the respective blocks are:

$$\begin{aligned} r_t &= W_r \cdot (\mu_r \odot x_t + (1 - \mu_r) \odot x_{t-1}) \\ k_t &= W_k \cdot (\mu_k \odot x_t + (1 - \mu_k) \odot x_{t-1}) \\ v_t &= W_v \cdot (\mu_v \odot x_t + (1 - \mu_v) \odot x_{t-1}) \\ r'_t &= W_{r'} \cdot (\mu_{r'} \odot x_t + (1 - \mu_{r'}) \odot x_{t-1}) \\ k'_t &= W_{k'} \cdot (\mu_{k'} \odot x_t + (1 - \mu_{k'}) \odot x_{t-1}) \end{aligned} \tag{17}$$

The *time-mixing block* takes a similar approach as the attention free transformers but in contrast to the AFTs it defines the pair-wise position biases as

$$w_{t,j} = -(t - j)w. \tag{18}$$

With that, the output vectors of the WKV block are calculated as

$$\text{wkv}_t = \frac{\sum_{j=1}^{t-1} \exp(k_j - (t - 1 - j)w_{t,j}) \odot v_j + \exp(k_j + u_t) \odot v_t}{\sum_{j=1}^{t-1} \exp(k_j - (t - 1 - j)w_{t,j}) + \exp(k_j + u_t)} \tag{19}$$

The introduction of the vector u_t is necessary to avoid any deterioration of the pair-wise position biases $w_{t,j}$. To get the output of the complete time mixing block, wkv_t is multiplied pointwise with the sigmoid of the receptance r_t and the result is multiplied by the output weight matrix.

$$\text{TimeMix}_t = W_o \cdot \big(\sigma(r_t) \odot \text{wkv}_t\big) \tag{20}$$

The receptance vector accumulates the information from prior time steps and can be seen as the hidden representation of all information until time t. Equation (19) shows the formulation that is used during training of the model. During inference, the calculation can be transformed into a recurrent formulation which is shown in equation (21) below:

$$\begin{aligned} \text{wkv}_t &= \frac{a_{t-1} + \exp(u_t + k_t) \odot v_t}{b_{t-1} + \exp u_t + k_t}, \\ a_0 &= 0, \\ b_0 &= 0, \\ a_t &= \exp(-w) \odot a_{t-1} + \exp(k_t) \odot v_t, \\ b_t &= \exp(-w) \odot b_{t-1} + \exp(k_t) \end{aligned} \tag{21}$$

In this formulation, a_t and b_t are updated recursively which gives the benefit of fast inference. This is the essence of the RKWV model: During training the formulation from equation (19) is used it can be computed in parallel. During inference, the formulation from equation (21) is used to have a recurrent prediction of the next output token.

After obtaining the output of the time-mixing block and a layer normalisation, the *channel-mixing block* "performs non-linear transformations across the feature dimensions, analogous to the feed-forward layers in Transformers" [18] by calculating

$$\text{ChannelMix}(x_t) = \sigma(W_{R'} \cdot x_t) \odot (W_{V'} \cdot \phi(W_{K'} \cdot x_t)) \tag{22}$$

with activation functions $\sigma(z) = \text{Sigmoid}(z)$ and $\phi(z) = \text{ReLU}(z)^2$. This gating of the information allows stable training and the non-linearity allows learning of complex features.

Combining the token shifting, the time and channel mixing, a complete block of a RWKV model as shown on the left side of figure 9 can be expressed with the following equations (23). For the ease of reading we neglect the layer normalisation and the RWKV-LM Head [18].

$$\begin{aligned} \text{ShiftedInput}_t &= \text{Shift}(x_t, x_{t-1}) \\ \text{TimeMixOutput}_t &= \text{TimeMix}(\text{ShiftedInput}_t) + \text{ShiftedInput}_t \\ \text{ChannelMixOutput}_t &= \text{ChannelMix}(\text{TimeMixOutput}_t) \\ \text{Output}_t &= \text{ShiftedInput}_t + \text{ChannelMixOutput}_t \end{aligned} \tag{23}$$

In real applications, multiple such blocks are stacked together. This allows the RWKV to be compatible with standard transformers without having a quadratic scaling in the sequence length.

C. State Space Model and its Extensions

The last model we want to present is the state space model which is completely transformer-free. State space models (SSMs) in a basic form are quite old as they date back to the 1960s when Rudolf Emil Kálmán published a paper in the field of control engineering [19]. These models linearly map a given 1-dimensional, continuous input $u(t) \in \mathbb{R}$ to an M-dimensional continuous output $y(t) \in \mathbb{R}$ via a hidden state $x(t) \in \mathbb{R}^N$. Usually, we expect the output dimension to be the same as the input dimension and hence set $M = 1$. Such an SSM is described by the system shown in equation (24).

$$\begin{aligned} \dot{x}(t) &= Ax(t) + Bu(t) \\ y(t) &= Cx(t) + Du(t) \end{aligned} \tag{24}$$

with $A \in \mathbb{R}^{N\times N}$, $B \in \mathbb{R}^{N\times 1}$, $C \in \mathbb{R}^{M\times N}$ and $D \in \mathbb{R}^{M\times 1}$ which will be the trainable parameters.

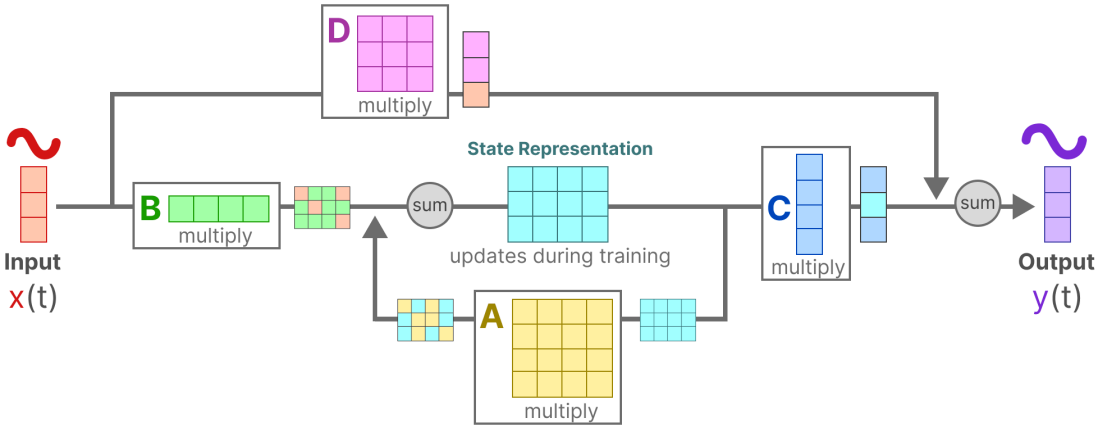

Fig. 10. Visualization of a standard SSM with a 3-dimensional input vector from [20].

As our input is usually discrete and not continuous, the system needs to be discretized. There exist multiple ways to do this. Gu et. al. [21] suggest to use the generalized bilinear transform from Arnold Tustin [22]. This transforms the matrices A and B into its discrete equivalents $\bar{A}$ and $\bar{B}$ in the following way using a step size Δ which defines how fine-grained the input is discretized:

$$\bar{A} = (I - \Delta/2 \cdot A)^{-1}(I + \Delta/2 \cdot A), \quad \bar{B} = (I - \Delta/2 \cdot A)^{-1}\Delta B \tag{25}$$

Another possible discretization is the zero-order hold (ZOH), which is shown in equation (26). This is the discretization rule Gu and Dao are using in their paper about the MAMBA architecture [23], which will be discussed later in this paragraph.

$$\bar{A} = \exp(\Delta A), \quad \bar{B} = (\Delta A)^{-1}(\exp(\Delta A) - I) \cdot \Delta B \tag{26}$$

Using a discretization and switching to the usual notation in machine learning for our inputs $x_t \in \mathbb{R}$, the hidden states $h_t \in \mathbb{R}^N$ and the outputs $y_t \in \mathbb{R}^M$, the discrete-time state-space model is given by

$$\begin{aligned} h_t &= \bar{A}h_{t-1} + \bar{B}x_t \\ y_t &= Ch_t \end{aligned} \tag{27}$$

Here we follow [24] and neglect the parameter D as the additional term Du is basically a skip connection and therefore its computation is simple.

The formulation in (27) corresponds to a usual recurrence, hence the parameters of the discrete SSM can be learned exactly as in RNNs. However, training the SSMs in a primitive, sequential way will not make use of the benefits of modern computing hardware. However, by unrolling the recurrence over time, one can see that the recurrent formulation can be transformed into a convolution. Let $h_{-1} = 0$, then the unrolled recurrence can be written as

$$h_0 = \bar{B}x_0, \quad h_1 = \bar{A}\bar{B}x_0 + \bar{B}x_1, \quad h_2 = \bar{A}^2\bar{B}x_0 + \bar{A}\bar{B}x_1 + \bar{B}x_2, \quad \ldots \tag{28}$$

$$y_0 = C\bar{B}x_0, \quad y_1 = C\bar{A}\bar{B}x_0 + C\bar{B}x_1, \quad y_2 = C\bar{A}^2\bar{B}x_0 + C\bar{A}\bar{B}x_1 + C\bar{B}x_2, \quad \ldots \tag{29}$$

This can be turned into the convolution

$$y = \bar{K} * x \tag{30}$$

with the convolution kernel $\bar{K} \in \mathbb{R}^L$ defined by

$$\bar{K} = (C\bar{B}, C\bar{A}\bar{B}, \ldots, C\bar{A}^{L-1}\bar{B}) \tag{31}$$

where $L \in \mathbb{R}$ is the length of the input sequence. The problem with this convolution formulation is that the matrix $\bar{A}$ is learned from data, hence the convolution kernel would need to be re-computed over and over again. In practice, without special restrictions on A, this is infeasible. In [21] the authors describe ways to overcome this issue. However, all of them are unstable or infeasible in practice.

In follow-up work Gu, Goel and Ré [24] present a new approach by introducing *Structured State Space Sequence Models (S4)*. Starting with an initialization of the structured state matrix A as the HiPPO matrix [25]

$$A_{nk} = -\begin{cases} (2n+1)^{\frac{1}{2}}(2k+1)^{\frac{1}{2}} & \text{if } n > k, \\ n+1 & \text{if } n = k, \\ 0 & \text{if } n < k. \end{cases} \tag{32}$$

and after some mathematical tricks - we leave it for the interested reader to go through the derivation in the paper - the authors present a stable Cauchy kernel which allows a computation in $\tilde{O}(N+L)$ with $O(N+L)$ memory usage. With this improvement, the S4 model and its flavors can achieve good results in a variety of applications including continuous data.

All SSM expect an one-dimensional input $x \in \mathbb{R}$ instead of $x \in \mathbb{R}^H$. To be able to process an input feature with a hidden dimension of H, the system in (27) needs to be broadcasted to the extra dimension of H. Every single feature $\left(x^{(h)}\right)_{h \in |H|}$ from the input takes its own copy of the system and therefore these parameters of the individual systems are learned independently. This means that there are overall H systems like (27) producing H output features which can be aggregated to the final output vector $y \in \mathbb{R}^H$. Unrolling this over time for an input sequence of length L results in an output sequence of shape $y \in \mathbb{R}^{L \times H}$. This is shown in Figure 10 where each single feature of the 3-dimensional input gets its own copy of B.

Another way of handling multi-dimensional input was developed by Smith, Warrington and Linderman [26]. They extend the S4 model to a *Structured State Space Model with a Scan (S5)*. They changed the architecture from having multiple stacked SSMs to having one single SSM capable of taking a multi-dimensional input and generating a multidimensional output from it. Doing so unfortunately makes the usage of the convolution computation impossible. Therefore the authors came up with the idea of using parallel scans instead, which gives the S5 layer the same complexity regarding the memory and runtime as the S4 layer and simultaneously reduces the architectural complexity.

However, for discrete and condensed data like text, the S4 and S5 models still lack performance. Gu and Dao [23] observed that these models are not able to focus only on relevant data, for example in text sequences. To overcome this problem, they extend the S5 model by a mechanism they call *selection*, turning these models into *Selective and Structured State Space Models for Sequences with a Scan (S6)*. Up until the S5 model, the presented SSM layers were linear time-invariant (LTI), which means that the parameters A, B, C and Δ are constant over time. The S6 model extends the S5 model by turning the system into a time-dependent one. Instead of taking the same matrices B, C and Δ for every time step, L different matrices are used with L being the length of the input sequence (Figure 11). All of these L matrices are then learned independently. To avoid a dramatic increase in runtime, Gu and Dao use the parallel scans algorithm as introduced in the S5 model [26] and suggest a hardware-aware algorithm, which allows the S6 model to have linear scaling in the sequence length.

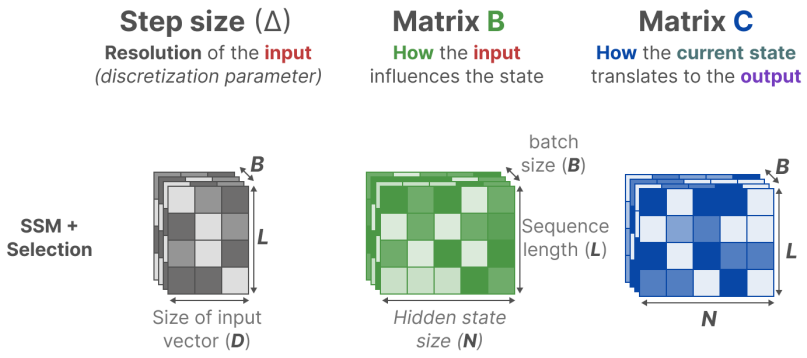

Fig. 11. Time Dependent Parameters in a S6 Model from [20].

Using the S6 layer as a basic building block, they propose a new foundation model called MAMBA (for details we refer to the paper [23]). The huge advantage of the MAMBA model compared to a transformer model is its computation time. The training can be done in parallel and the inference is done in an auto-regressive fashion making it a potential competitor to the transformer model.

D. Comparison of Architectures

In this chapter, we want to summarize our findings by giving an overview of the runtime and performance of the introduced architectures. Again, n is the given sequence length and d is the dimension of the model.

TABLE I
COMPARISON OF RUNTIME AND SPACE COMPLEXITY ACROSS VARIOUS MODELS FROM [15], [23].

Model	*Time*	*Space*
Transformer	$O(n^2 d)$	$O(n^2 + nd)$
Reformer	$O(n \log nd)$	$O(n \log n + nd)$
Performer	$O(nd^2 \log d)$	$O(nd \log d + d^2 \log d)$
AFT	$O(n^2 d)$	$O(nd)$
RWKV	$O(nd)$	$O(d)$
Mamba	$O(nd)$	$O(d)$

Table I shows the runtime and the space requirements of all presented architectures. We can see that attention-free architectures have the advantage of fewer space requirements compared to the standard transformer. However, work of Zhai et. al. [14] also shows that this comes with the drawback of having a reduction in performance.

From all presented architectures, the RWKV and the Mamba model have the least time and space complexity, making them potentially far more efficient than transformers. However, Peng et al. [15] highlight several caveats in the RWKV model, namely a potential restriction in modeling minutiae information due to having only one single hidden state vector where the context information needs to be stored in as well as the necessity of good prompt engineering.

Mamba faces similar problems. As it also compresses information, it may lose track of relevant context over time. This is in contrast to the transformer as it can arbitrarily and at any point in time look back to every single token in a sequence history. However, there is currently too not enough data to finally judge how Mamba behaves in a large-scale, real-world application. [23].

VII. CONCLUSION

In conclusion, Transformers remain the dominant architecture for NLP tasks due to their unmatched ability to model long-range dependencies and their flexibility across various applications. However, their high computational and memory requirements pose challenges, particularly for resource-constrained environments. For such scenarios, Reformer and Performer provide viable alternatives. Reformer excels in processing long sequences efficiently through LSH-based sparse attention, while Performer's kernel-based approximation offers linear complexity, making it ideal for real-time tasks.

Beyond Transformers, attention-free architectures like RWKV and MAMBA demonstrate the potential for handling sequence modeling with reduced complexity. RWKV bridges the advantages of recurrent models and Transformers by enabling parallel training and efficient inference, while MAMBA's structured state space design shows promise for auto-regressive tasks. These architectures offer compelling options for applications where computational efficiency or interpretability is paramount.

As the field progresses, the choice of architecture should align with task-specific requirements, such as sequence length, resource availability, and real-time processing needs. By leveraging the strengths of these emerging models, NLP applications can achieve greater scalability and efficiency, opening doors to new possibilities in diverse domains.

REFERENCES

[1] J. Weizenbaum, "ELIZA—a computer program for the study of natural language communication between man and machine," *Communications of the ACM*, vol. 9, no. 1, pp. 36–45, 1966.

[2] T. Winograd, *Understanding Natural Language*. Academic Press, 1972.

[3] Y. Wilks, "Preference semantics," in *Formal Semantics of Natural Language*, E. L. Keenan, Ed. Cambridge University Press, 1975, pp. 329–348.

[4] N. Chomsky, *Syntactic Structures*. Mouton, 1957.

[5] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2013, arXiv preprint arXiv:1301.3781.

[6] J. Pennington, R. Socher, and C. D. Manning, "GloVe: Global vectors for word representation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, 2014, pp. 1532–1543.

[7] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[8] Y. Kim, "Convolutional neural networks for sentence classification," in *EMNLP*, 2014, pp. 1746–1751.

[9] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015, arXiv preprint arXiv:1409.0473.

[10] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, vol. 30. Curran Associates, Inc., 2017, pp. 5998–6008.

[11] N. Kitaev, Ł. Kaiser, and A. Levskaya, "Reformer: The efficient transformer," in *Proceedings of the 8th International Conference on Learning Representations (ICLR)*, 2020, arXiv preprint arXiv:2001.04451.

[12] A. N. Gomez, M. Ren, R. Urtasun, and R. B. Grosse, "The reversible residual network: Backpropagation without storing activations," in *Advances in Neural Information Processing Systems*, vol. 30. Curran Associates, Inc., 2017, pp. 2214–2224.

[13] K. Choromanski, V. Likhosherstov, D. Dohan, X. Song, A. Gane, T. Sarlos, P. Hawkins, J. Q. Davis, A. Mohiuddin, Ł. Kaiser *et al.*, "Rethinking attention with performers," in *Proceedings of the 9th International Conference on Learning Representations (ICLR)*, 2021, arXiv preprint arXiv:2009.14794.

[14] S. Zhai, W. Talbott, N. Srivastava, C. Huang, H. Goh, R. Zhang, and J. Susskind, "An attention free transformer," 2021. [Online]. Available: https://arxiv.org/abs/2105.14103

[15] B. Peng, E. Alcaide, Q. Anthony, A. Albalak, S. Arcadinho, S. Biderman, H. Cao, X. Cheng, M. Chung, M. Grella, K. K. GV, X. He, H. Hou, J. Lin, P. Kazienko, J. Kocon, J. Kong, B. Koptyra, H. Lau, K. S. I. Mantri, F. Mom, A. Saito, G. Song, X. Tang, B. Wang, J. S. Wind, S. Wozniak, R. Zhang, Z. Zhang, Q. Zhao, P. Zhou, Q. Zhou, J. Zhu, and R.-J. Zhu, "Rwkv: Reinventing rnns for the transformer era," 2023. [Online]. Available: https://arxiv.org/abs/2305.13048

[16] S. Hochreiter and J. Schmidhuber, "Long short-term memory." *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[17] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," 2014. [Online]. Available: https://arxiv.org/abs/1412.3555

[18] A. Datta, "The evolution of rwkv: Advancements in efficient language modeling," 2024. [Online]. Available: https://arxiv.org/abs/2411.02795

[19] R. E. Kálmán, "A new approach to linear filtering and prediction problems," *Transactions of the ASME–Journal of Basic Engineering*, vol. 82, pp. 35–45, 1960.

[20] M. Grootendorst. (2024) A visual guide to mamba and state space models. Accessed: Nov. 15, 2024. [Online]. Available: https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mamba-and-state

[21] A. Gu, I. Johnson, K. Goel, K. Saab, T. Dao, A. Rudra, and C. Ré, "Combining recurrent, convolutional, and continuous-time models with linear state-space layers," 2021. [Online]. Available: https://arxiv.org/abs/2110.13985

[22] A. Tustin, "A method of analysing the behaviour of linear systems in terms of time series," *Journal of the Institution of Electrical Engineers-Part IIA: Automatic Regulators and Servo Mechanisms*, vol. 94(1), pp. 130–142, 1947.

[23] A. Gu and T. Dao, "Mamba: Linear-time sequence modeling with selective state spaces," 2024. [Online]. Available: https://arxiv.org/abs/2312.00752

[24] A. Gu, K. Goel, and C. Ré, "Efficiently modeling long sequences with structured state spaces," 2022. [Online]. Available: https://arxiv.org/abs/2111.00396

[25] A. Gu, T. Dao, S. Ermon, A. Rudra, and C. Re, "Hippo: Recurrent memory with optimal polynomial projections," 2020. [Online]. Available: https://arxiv.org/abs/2008.07669

[26] J. T. H. Smith, A. Warrington, and S. W. Linderman, "Simplified state space layers for sequence modeling," 2023. [Online]. Available: https://arxiv.org/abs/2208.04933