Scaleformer : a scalable transformer with linear complexity and relative positional encoding

Benoit Favier[1] and Walter Dal'Maz Silva[2]

[1]Phealing
[2]Unaffiliated

February 19, 2022

Abstract

To overcome the quadratic complexity with sequence length of the original transformer, some previous works proposed a kernelized attention mechanism which can scale linearly depending on operation orders. Other works proposed to change the way the position of each token is encoded so that the model depends on relative distance between tokens instead of absolute position. In this work we propose a novel algorithm to combine kernelized attention with relative positional encoding while still scaling linearly in complexity.

1 Introduction

Vaswani et al. [11] introduced the transformers, a novel architecture for sequence-to-sequence tasks, tackling new heights of problem complexity. The improvements were due to the non sequential nature of its training process which allowed better parallelization and thus fast training of big models, and due to improved long term dependencies thanks to the intrinsic good gradient flows of the attention mechanism. However the original transformer presents some drawbacks. It has a quadratic complexity with sequences length. And it's absolute position encoding is detrimental to it's extrapolation to new sequence lengths.

Ever since, significant efforts have been made to alleviate these problems. These improvements have opened the path to the application of transformers to image analysis where scalability and positional invariance are essential. However most proposed architecture we are aware of only alleviated parts of the issues, by being incompatible with relative positional encoding, having no scheme for masked attention - thus being restrained to encoder-only models, or being complex to use/implement - needing custom operations programmed in CUDA or introducing stochastic methods.

In the present work we propose a complete encoder-decoder transformer model that scales linearly with sequence lengths, by putting together ideas developed across several works [2, 4, 6, 9–11]. It remains compatible with relative positional encoding, and can be implemented with usual functions of neural network frameworks without requiring custom CUDA code.

2 Background

2.1 The original multi-head attention mechanism

The scaled dot-product attention proposed by Vaswani et al. [11] transforms the vectorial embedding $\vec{Y}_i$ of a token, as a function of a sequence variable in size of other tokens $\vec{X}_j$. Where $\vec{Y}_i$ and $\vec{X}_j$ are all vectors of size D. A key $\vec{K}_j$ and value $\vec{V}_j$ are attributed to each vector $\vec{X}_j$, and query $\vec{Q}_i$ is attributed to $\vec{Y}_i$. The query keys and values are obtained by linear projection from dimension D to d using three matrices of learnable parameters. The transformed vector $\overrightarrow{y_i'}$ is a weighted sum of the $\vec{V}_j$. The weights are scores of matching between the query $\vec{Q}_i$ and the keys $\vec{K}_j$, calculated as the dot product between the two vectors. The weights are also *softmaxed* to sum up to 1.

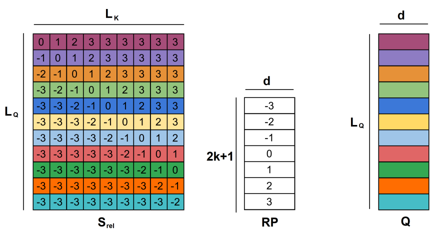


Figure 1: S_{rel} calculation

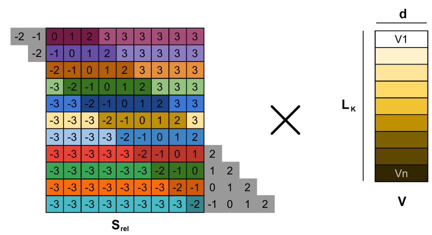


Figure 2: A^{rel} naive calculation

The transformation of L_Q vectors $\vec{Y_i}$ as a function of L_K vectors $\vec{X_j}$ can be efficiently computed with matrix multiplications:

$$A = \text{softmax}\left(\frac{Q \times K^T}{\sqrt{d}}\right) \times V \quad (1)$$

With Q a matrix of shape (L_Q, d), K a matrix of shape (L_K, d) and V a matrix of shape (L_K, d). The $\sqrt{d}$ at the denominator is a scaling factor used to avoid saturation in the exponential terms of the softmax function.

The multi-head attention performs h different projections into spaces of dimension $d = D/h$. The resulting vector $\vec{Y_i}'$ is the concatenation of the h vectors $\overrightarrow{y_i'}$ obtained. Thus the embedding dimension is preserved. Using multiple heads was found beneficial by the authors over using a single head of dimension $d = D$.

During training, the cross entropy of the n^{th} predicted token is calculated assuming all previous tokens have been generated correctly. This enables to parallelize training completely as there is no recurrence in the calculation process. However as the n^{th} token should not depend of the following tokens, the cells in the upper right corner of the score matrix are set to $-\infty$ such that after the softmax they are equal to 0, and the rows still sums up to 1.

2.2 Improving scalability

The original attention mechanism requires the computation of a score matrix $Q \times K^T$ of shape (L_Q, L_K), with complexity $O(L_Q d L_K)$. If the query and key sequence lengths are multiplied by two, then the memory used and computation time are multiplied by 4. To improve the scalability of the transformer with sequence length, several axis of research have been explored.

Kitaev et al. [7] proposed the Reformer's architecture, which uses an hash-bucketting algorithm to reduce the complexity of the original multi head attention operation from $O(L^2)$

to $O(L\log(L))$.

Dai et al. [3] proposed the Transformer-XL's architecture, which cuts the sequence in segments of length L. The model predicts each stage of the current segment as a function of the previous and current segment. All the segments are computed sequentially with a recurrence mechanism. The complexity is linear with sequence length, but the computation cannot be completely parallelized due to the recurrence mechanism.

Other publications explored using a sparse attention matrix, such as the Longformer by Beltagy et al. [1] and the Big Bird model by Zaheer et al. [13]. As each token attends to a fixed number of all other tokens, the scalability is improved. These sparse attention models however require custom operations implemented in CUDA.

Some other works propose to modify the attention mechanism so that it's complexity scales linearly with sequence length. The Linformer by Wang et al. [12] projects the key and values onto a smaller sequence length dimension with matrix multiplication. It cannot however generalize to sequences longer than during training, as the weights of the projection for such tokens would be undefined.

Shen et al. [10] proposed to replace the softmax attention score. $A = \text{softmax}\left(\frac{Q\times K^T}{\sqrt{d}}\right) \times V$ is changed into $A = \rho(Q) \times \rho(K)^T \times V$. With ρ the softmax function along the embedding dimension. Thanks to matrix multiplication commutativity, the order of the operations can be chosen. If Q, K and V are of shape (L_Q, d), (L_K, d) and (L_K, d) respectively, the complexity of $(\rho(Q) \times \rho(K)^T) \times V$ is $O(L_Q \times d \times L_K)$ whereas the complexity of $\rho(Q) \times (\rho(K)^T \times V)$ is $O\left(max(L_Q, L_k) \times d^2\right)$. The right-side-first operation is linear in complexity with sequence length. The shape of the intermediate result matrix is also changed, allowing to scale better in memory requirements as well. The original *softmaxed* attention score matrix was giving rows of positive scores that sum to 1. With this change the elements of the score matrix remain positive as $\rho(Q)$ and $\rho(K)^T$ are matrices of positive values, but the rows of the score matrix does not sum up to 1. This work also does not give a linear complexity formulation for masked attention. If the right-side-first scheme is adopted, the attention score matrix $\rho(Q) \times \rho(K)^T$ is never explicitly computed, and can't be masked.

Building on this idea of commutative attention function proposed by [10], Katharopoulos et al. [6] introduced their kernerlized attention function as:

$$A = \frac{\phi(Q) \times \phi(K)^T}{\sum_j (\phi(Q) \times \phi(K)^T)} \times V \quad (2)$$

The function ϕ is applied element-wise and can be any positive function, for example $\phi(x) = elu(x) + 1$. This attention is row-wise normalized so that all rows of the score matrix are sets of positive weights adding up to one. This preserves the objective of the original softmaxed attention scores, while allowing to perform operations in an optimal order.

The Performer by Choromanski et al. [2] exploits the same idea of a kernelized attention introduced by Katharopoulos et al. [6], with an algorithm that better approximates softmaxed attention. Most importantly they also give in annex a prefix sum algorithm to perform operations in the right-side-first order while giving the same result as masked left-side-first operation. Althought they give no insight in their paper as how the operation could be implemented without custom CUDA code.

In this work we will explicit an implementation of the right-side-first masked operation, with usual functions from neural network frameworks, that remains linear in complexity.

2.3 Alternative positional encodings

The original multi-head attention operation introduced by Vaswani et al. [11] was intrinsically invariant by token order permutation. As token position was an important information for machine translation models, they encoded the global position of each token in their embedding. Since then, some modified attention mechanisms, that depend on relative tokens position, have been proposed.

Shaw et al. [9] explored modifying the attention mechanism so that it depends on the relative distance between tokens. A second score matrix that is function of the query and the query/key relative distance is added to the original score matrix. $A = softmax\left(\frac{Q \times K^T}{\sqrt{d}}\right) \times V$ becomes $A = \left(\frac{Q \times K^T + S_{rel}}{\sqrt{d}}\right) \times V$ with S_{rel} of shape (L_Q, L_K) defined as $S_{relij} = \vec{Q_i} \cdot \vec{RP}_{clip(i-j,-k,k)}$. Where k is the attention horizon length and $\vec{RP}_n$ is one of $2k+1$ relative positional embedding, vectors of size d. Shaw et al. [9] and Huang et al. [5] observed that introducing this attention scheme improved performances. The naive calculation of this term however has a complexity of $O(L_Q L_K d)$. No algorithm was provided to linearize the complexity.

More recently Liutkus et al. [8]

gives a stochastic positional encoding that is linear in complexity with regards to sequence length. However the implementation is complex and its stochastic nature requires that the operations be repeated several times in parallel.

Horn et al. [4] noted that the term $S^{rel} \times V$ can be computed with linear complexity for the case where $RP_{-k} = RP_k$. However this is restraining as the model can't make the difference between tokens before the attention horizon or after.

In this work we will show that the computation of $S^{rel} \times V$ can also be done with linear complexity, without concession.

3 Linear complexity masked attention

In this section we will detail the prefix sum algorithm proposed by Choromanski et al. [2] for masked kernelized attention, and give an implementation with usual functions of neural network frameworks. The masked kernelized attention is defined as:

$$A^{masked} = \text{masked}\left(\phi(Q) \times \phi(K)^T\right) \times V \tag{3}$$

In this expression, masked being the operation that set all cells above the diagonal to 0 in a matrix. The naive implementation of this operation has complexity $O(L_Q L_K d)$.

We can change the complexity using the operation proposed by Choromanski et al. [2]. We will derive its formulation here. To start with it, A^{masked} is expressed as

$$A^{masked} = S^{masked} \times V \tag{4}$$

which in summation form is expressed as

$$A^{masked}_{ij} = \sum_k V_{kj} \times S^{masked}_{ik} \tag{5}$$

and the elements of S^{masked} are defined as:

$$S^{masked}_{ik} = \begin{cases} \sum_l \left(\phi(Q)_{il} \times \phi(K)_{kl}\right) & \text{if } k \leq i \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

Putting these elements together leads to

$$A^{masked}_{ij} = \sum_{k=1}^{i} \left(V_{kj} \times \sum_l \left(\phi(Q)_{il} \times \phi(K)_{kl}\right)\right) \tag{7}$$

which can be reworked as

$$A^{masked}_{ij} = \sum_l \left(\phi(Q)_{il} \times \sum_{k=1}^{i} \left(V_{kj} \times \phi(K)_{kl}\right)\right) \tag{8}$$

In this work we make use of these ideas to implement the calculation of A^{masked} with complexity $O(max(L_Q, L_K) \times d^2)$ without custom CUDA code as per the algorithm 1. With:

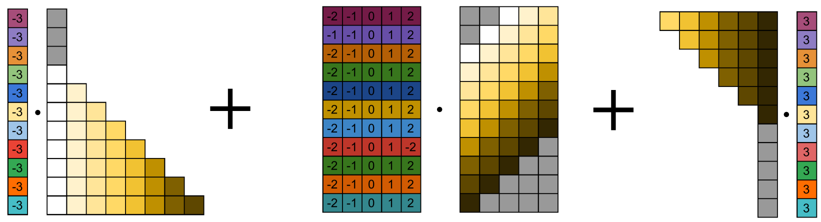


Figure 3: A_{rel} linear complexity calculation

- $align(tensor, n, dim)$ the function that truncates or repeats the last value so that $tensor$ has length n along the given dimension
- $cumsum(tensor, dim)$ the function that returns the cumulated sum along the given dimension

4 Linear complexity RPE

In this section we will detail how the relative positional encoding proposed by Shaw et al. [9] can in fact be computed with linear complexity. The S_{rel} matrix of scores is computed as $S_{relij} = \overrightarrow{Q_i} \cdot \overrightarrow{RP}_{clip(i-j,-k,k)}$.

In figure 1 the colors represent the index of the query and the number the index of the relative position.

The relative positional encoding's term of the attention is calculated as $A_{rel} = S_{rel} \times V$. Each row of the A_{rel} matrix is a weighted sum of the value vectors V_i. Each row of S_{rel} is a set of weights.

One can observe in figure 2 that some weights are repeated several times in the S_{rel} matrix. Calculating the whole matrix can be avoided by instead calculating all possible weights only once, with complexity $O\left(L_Q \times d \times (2k+1)\right)$. As illustrated in figure 3, the matrix multiplication can then be replaced by a sum of three terms. The embedding dimension of size d is not represented here. Instead the horizontal axis in the figure represents all terms that must be summed. The grey squares represent some zero-padding. The first term is a cumulated sum of the value vectors that is weighted by the before-horizon set of weights (complexity $O(max(L_Q, L_K))$). The second term is a weighed sum of a moving window of the value vectors. Where the weights are the central diagonal of the S_{rel} matrix. (complexity $O(L_Q \times (2k-1) \times d)$). The last term is similar to the first term, but

Algorithm 1: calculation of A^{masked} with linear complexity

Input: $\phi(Q)$ query vectors, tensor of shape (L_Q, d)
$\phi(K)$ key vectors, tensor of shape (L_K, d)
V value vectors, tensor of shape (L_K, d)
Data: Expanded tensor of shape (L_K, d, d)
Summed tensor of shape (L_K, d, d)
Aligned tensor of shape (L_Q, d, d)
Result: A^{masked} tensor of shape (L_Q, d)
$\text{Expanded}_{kjl} := V_{kj} \times \phi(K)_{kl}$
$\text{Summed} := cumsum\,(\text{Expanded}, \text{dim=0})$
$\text{Aligned} := align(\text{Right}, L_Q, \text{dim=0})$
$A^{masked}_{ij} := \sum_k \left(\phi(Q)_{ik} \times \text{Aligned}_{ikj}\right)$

for the after-horizon set of weights (complexity $O(max(L_Q, L_K))$). The implementation is given in algorithm 2. The algorithm can be applied for masked attention by settings all vectors of the RP matrix with strictly positive indexes to $\vec{0}$.

5 The Scaleformer

The proposed model replaces the scaled-dot-product-attention by a kernelized attention with RPE. Following the observations of Shaw et al. [9] that accumulating absolute positional encoding with relative positional encoding yield no benefits, the positional encoding is also removed.

In this work we have chosen the following formulation, with $\phi(x) = exp(x)$.

$$A = \frac{\left(\phi(Q) \times \phi(K^T) + S^{rel}\right)}{\sum_j \left(\phi(Q) \times \phi(K^T) + S^{rel}\right)} \times V \tag{9}$$

This is essentially a combination of two terms: the kernelized attention proposed by Katharopoulos et al. [6], and the relative positional encoding proposed by Shaw et al. [9]. The left term is the score matrix of shape (L_Q, L_K), with a denominator which scales all rows so that they sum to 1.

For the linear complexity implementation, the multiplication must be distributed as:

$$A = \frac{\left(\phi(Q) \times \phi(K^T) \times V\right) + \left(S^{rel} \times V\right)}{\sum_j \left(\phi(Q) \times \phi(K^T)\right) + \sum_j \left(S^{rel}\right)} \tag{10}$$

The denominator can be easily calculated by applying the linear complexity algorithms with V re-

Algorithm 2: calculation of A^{rel} with linear complexity

Input: $\phi(Q)$ query vectors, tensor of shape (L_Q, d)
RP relative position embeddings, tensor of shape $(2k+1, d)$
V value vectors, tensor of shape (L_K, d)

Data: W tensor of shape $(L_Q, 2k+1)$
$n_{before} := min(max(0, L_Q - k), L_K)$, scalar
$p_{before} := min(k, L_Q)$, scalar
$Cumulated$ tensor of shape (L_Q, d)
$Summed$ tensor of shape (d)
$Rcumulated$ tensor of shape (L_Q, d)
$W^{horizon}$ tensor of shape $(L_Q, 2k-1)$
$V^{horizon}$ tensor of shape $(L_Q, 2k-1, d)$
$n_{after} := min(L_Q + k, L_K)$, scalar
$p_{after} := max(0, Lq - max(0, Lk - k))$, scalar

Result: A_{rel} tensor of shape (L_Q, d)

$W := \phi(Q) \times RP^T$
$Cumulated := cumsum(V, dim = 0)$
$W_i^{before} := W_{i,0}$
$$V_{ij}^{before} := \begin{cases} 0 & \text{if } i < p_{before} \\ Cumulated_{i-p_{before},j} & otherwise \end{cases}$$
$A_{ij}^{before} := W_i^{before} \times V_{ij}^{before}$
$W_{ij}^{horizon} := W_{i+1,j}$
$$V_{ijl}^{horizon} := \begin{cases} V_{i-k+1+j,l} & \text{if } 0 \leq (i-k+1+j) < L_K \\ 0 & \text{otherwise} \end{cases}$$
$A_{il}^{horizon} := \sum_j \left(W_{ij}^{horizon} \times V_{ijl}^{horizon}\right)$
$Summed_j := \sum_i (V_{ij})$
$$Rcumulated_{ij} := \begin{cases} Summed_j & \text{if } i = 0 \\ Summed_j - Cumulated_{i-1,j} & otherwise \end{cases}$$
$W_i^{after} := W_{i,2k}$
$$V_{ij}^{after} := \begin{cases} Rcumulated_{i+k-1,j} & \text{if } i < n_{after} \\ 0 & otherwise \end{cases}$$
$A_{ij}^{after} := W_i^{after} \times V_{ij}^{after}$
$A_{ij}^{rel} := A_{ij}^{before} + A_{ij}^{horizon} + A_{ij}^{after}$

Figure 4: A_{rel} calculation runtimes for $d = 64$ and $k = 16$

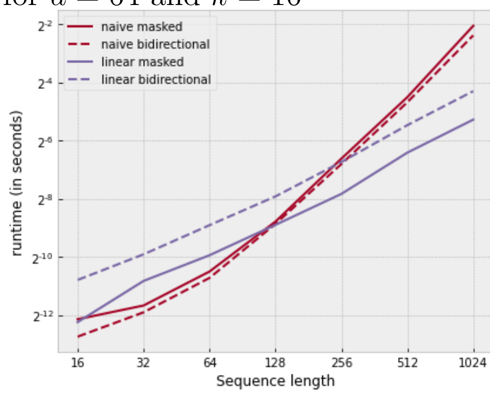
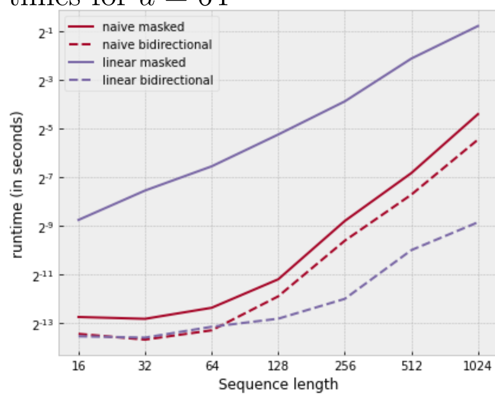


Figure 5: kernelized attention runtimes for $d = 64$



placed by a matrix of shape $(L_K, 1)$ full of 1, or by summing the rows of the score matrix for quadratic complexity algorithm.

The linear complexity algorithm is a trade-of of quadratic complexity with sequence length for quadratic complexity with projection dimension. Increasing expressiveness power of the model can however be done efficiently by increasing the number of heads. As both the naive and linear algorithm have linear complexity with the number of attention heads.

6 Results

The various algorithms have been timed on CPU and runtimes have been plotted against sequences length in log2-log2 scale. Algorithms that scale linearly with sequence length have a slope of 1, while algorithms that scale quadratically with sequence length have a slope of 2. The Figure 4, shows the timings of the A_{rel} matrix calculation.

On Figure 5 the kernelized attention algorithms have been timed. We can observe that although our algorithm for masked attention has a linear complexity, it is still often slower than the quadratic complexity algorithm. The bottleneck in our implementation was the cumulated sum operation.

7 Conclusion

In the present work an algorithm to compute Shaw et al. [9] relative positional embedding with linear complexity has been presented. An implementation of Choromanski et al.

[2] prefix sum algorithm that doesn't requires custom CUDA code - while maintaining linear complexity - was also presented. These two elements allowed to define a kernelized attention function with relative positional encoding, that can be computed with linear complexity with regards to sequence length. This opens the path to sequence to sequence models that can be applied to massive document sizes. For example chat bots trained on small sequence length, that can generalize to long conversations thanks to relative positional encoding, and maintain fast evaluation times thanks to linear complexity.

Table of symbols

$\boldsymbol{A_{rel}}$	the result of the dot product attention operation, tensor of shape (L_Q, d)
$\boldsymbol{d}$	the projection dimension
$\boldsymbol{D}$	the embedding dimension
$\boldsymbol{h}$	the number of heads
$\boldsymbol{k}$	the radius of the attention horizon
$\boldsymbol{K}$	the keys linked to the attended sequence, tensor of shape (L_K, d)
$\boldsymbol{L_K}$	the sequence length of the attended sequence
$\boldsymbol{L_Q}$	the sequence length of the transformed sequence
$\boldsymbol{Q}$	the query linked to the sequence to transform, tensor of shape (L_Q, d)
$\boldsymbol{S_{rel}}$	The matrix of relative position scores, tensor of shape (L_Q, L_K)
$\boldsymbol{V}$	the values linked to the attended sequence, tensor of shape (L_K, d)

References

[1] Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer, 2020. URL https://arxiv.org/abs/2004.05150.

[2] Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, David Belanger, Lucy Colwell, and Adrian Weller. Rethinking attention with performers, 2021. URL https://arxiv.org/abs/2009.14794.

[3] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context, 2019. URL https://arxiv.org/abs/1901.02860.

[4] Max Horn, Kumar Shridhar, Elrich Groenewald, and Philipp F. M. Baumann. Translational equivariance in kernelizable attention, 2021. URL https://arxiv.org/abs/2102.07680.

[5] Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Noam Shazeer, Ian Simon, Curtis Hawthorne, Andrew M. Dai, Matthew D. Hoffman, Monica Dinculescu, and Douglas Eck. Music transformer, 2018. URL https://arxiv.org/abs/1809.04281.

[6] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention, 2020. URL https://arxiv.org/abs/2006.16236.

[7] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer, 2020. URL https://arxiv.org/abs/2001.04451.

[8] Antoine Liutkus, Ondřej Cífka, Shih-Lun Wu, Umut Şimşekli, Yi-Hsuan Yang, and Gaël Richard. Relative positional encoding for transformers with linear complexity, 2021. URL https://arxiv.org/abs/2105.08399.

[9] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations, 2018. URL https://arxiv.org/abs/1803.02155.

[10] Zhuoran Shen, Mingyuan Zhang, Haiyu Zhao, Shuai Yi, and Hongsheng Li. Efficient attention: Attention with linear complexities, 2020. URL https://arxiv.org/abs/1812.01243.

[11] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017. URL https://arxiv.org/abs/1706.03762.

[12] Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity, 2020. URL https://arxiv.org/abs/2006.04768.

[13] Manzil Zaheer, Guru Guruganesh, Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big bird: Transformers for longer sequences, 2021. URL https://arxiv.org/abs/2007.14062.