INATTENTION: LINEAR CONTEXT SCALING FOR TRANSFORMERS

JOSEPH D. EISNER

ABSTRACT. VRAM requirements for transformer models scale quadratically with context length due to the self-attention mechanism. In this paper we modify the decoder-only transformer, replacing self-attention with InAttention, which scales linearly with context length during inference by having tokens attend only to initial states. Benchmarking shows that InAttention significantly reduces VRAM usage during inference, enabling handling of long sequences on consumer GPUs. We corroborate that fine-tuning extends context length efficiently, improving performance on long sequences without high training costs. InAttention offers a scalable solution for long-range dependencies in transformer models, paving the way for further optimization.

1. BACKGROUND AND INTRODUCTION

Decoder-based transformer stacks [16] have demonstrated syntactic and semantic understanding of language and other time-series data, achieving state-of-the-art few-shot [3] performance across nearly any natural language task. They exhibit predictable scaling laws with respect to number of parameters and the amount of training data they ingest: bigger is better and we generally know by how much [8].

These *Foundation Models* seem to benefit from extended context length but the self-attention mechanism at the heart of the transformer scales quadratically with context length – making training and inference on extremely long queries cost prohibitive. One way to make long queries cheaper is to make the attention mechanism sparse, not allowing tokens to attend every token before them, but merely a subset.

Mohtashami and Jaggi [11] give an excellent summary[1] of sparse attention techniques:

> ...For example, Child et al. [6] limit the attention to a local window around each token, while BigBird additionally suggests attending to a random subset of previous tokens as well as several globally accessible tokens [18]. Longformer [2] further introduces dilated sliding window patterns to increase attention's receptive field and manually picks the window sizes for each layer. Linformer [17] uses a low-rank approximation of the attention matrix while Performer [7] uses a non-softmax kernel to obtain a more efficient implementation. Reformer [9] uses locality-sensitive-hashing (LSH) to retrieve the closest key vectors which should account for the highest scores in the attention matrix. Combiner [13] utilizes a hierarchical attention mechanism and heuristic reduction techniques, such as max-pooling, to derive key and query vectors for input blocks...
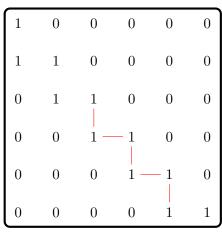
---

[1]Citations updated to match our bibliography.

while their own approach involves inserting special *landmark* tokens to stand in for consecutive blocks – allowing models to restrict their attention to blocks which contain at least one high scoring token.

Perhaps the most human-interpretable form of sparse attention is *sliding window* attention (e.g. Mistral's [1]), which uses a lower-diagonal banded matrix as the attention mask. This bifurcates the notion of "context length": there is the *literal context*, the length of the band, and there is an *effective context* which is the length of the band multiplied by the number of layers, see Figure 1. It is difficult to assess how well the effective context is truly utilized by the model.

Dense Attention Mask   *Sliding Window Mask*

FIGURE 1. Right: A sliding window mask with literal context length of 1. Outlined in red is the path information might take in a 3-layer transformer model giving an effective context length of 3.

In this paper we propose a technique for which **dense** attention scales linearly with context length. We do this by having the hidden states at each layer attend the initial states of a query, rather than themselves, inspiring the name *InAttention*. This has two major downstream effects on inference:

1. The attention matrix becomes an attention vector, since we do not need to calculate any hidden states except the ultimate one. For long queries, the attention matrix represents an enormous overhead, potentially dwarfing the size of the model being used. Consequently, we dramatically reduce the maximum VRAM needed for the initial next-token prediction.
2. After the initial next-token prediction, it is common practice to cache the hidden states at each layer so they do not need to be recomputed. This prevents us from needing to spin up the attention matrix again but it still represents substantial overhead. With InAttention, since we only attend initial states, we do not need to do this – freeing up VRAM even downstream of the initial surge from (1).

Our experiments suggest that there is a small but real decrease in model capability (measured by training loss) by using InAttention, which we will describe in Section 3.2. However, we consider this a small price to pay for the gains and suggest the capability be bolstered by allocating some of the freed-up VRAM to additional model parameters, see Figure 9.

While InAttention allows for inference on extremely long queries, it does not make it any cheaper to train the model, where we will still want to compute losses for every token prediction in a training batch and so the cost is still quadratic. To partially address this, we corroborate that models can be cheaply finetuned to extend their context length (as suggested in e.g. [4], [15], and [5]), our analysis is in Section 4.1.

Our hope is that further efficiencies will be found which, paired with InAttention, will allow for both cheaper training and inference on long queries. While InAttention on its own does not solve training, it does essentially solve inference, and we hope this is a major step towards running exceedingly long queries on consumer hardware.

1.1. **Why Compute Scales Quadratically With Context Length.**

We refer to the sequence of vectors connected via residual connections as a *residual tower*. Our perspective is that this sequence represents a vector "morphing" or "mutating" as it ascends the transformer stack.

During the self-attention step of the transformer each token t must attend a substantial number of other tokens $\sigma(t)$, depending on the mask, and this number grows linearly as the number of tokens T is increased, $\sigma(t) \sim T$. So the total number of attention pairs we must compute is $\sum_t \sigma(t) \sim \sum_t T \sim T^2$.

In the case of decoder-based transformers, which use a causal mask, each token must attend itself and every token which comes before it, yielding $\frac{T^2+T}{2}$ attention pairs in each self-attention mechanism given a context length of T tokens.

Sparse attention addresses quadratic scaling by bounding $\sigma(t) < C, C \in \mathbb{N}$, so the number of attention computations becomes $\sum_t \sigma(t) \leq \sum_t C = CT$, thus achieving linear scaling with context length. In the next subsection we will share some intuition for InAttention but a mathematically simple summary is this: Where sparse attention sees the expression $\sum_t \sigma(t)$ and seeks to bound it by controlling σ, InAttention instead looks to remove the sum.

2. InAttention: Attending Initial Tokens Only

2.1. **Intuition for InAttention.**

When the token-vectors work their way to the "top" of the transformer stack, each represents the model's latent approximation of a prediction for the next token. For instance, given the phrase "Every planet deserves a moon", the vector originally representing "Every" morphs into (a latent approximation of) the models prediction for the token following "Every", perhaps "dog", anticipating "Every dog has its day". The vector originally representing "planet" likewise morphs into the model's prediction for the token following "planet", perhaps "in" anticipating "Every planet in the solar system". It is interesting that in predicting the token following "planet" we have the

vector attending the (typically) false prediction "dog" (or a latent approximation thereof). Reference Figure 2.
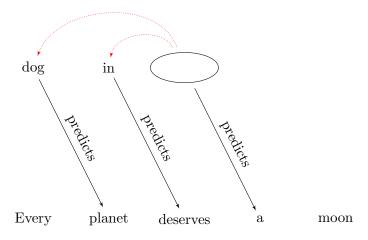

FIGURE 2. Next Token Game Demonstration

It is conceivable (and appears to be the case, see Section 3.2) that the latent representation which predicts "dog" also contains some partial work towards subsequent tokens, which the model can utilize. But it also seems plausible that many of the features in that prediction are not useful, and much of the work is immediately depreciated with the revealing of the true next token, "planet".

During inference, the final layer of vectors is mostly uninteresting, since the linear decoder layer does not contain a self-attention mechanism. The aforementioned predictions "dog" and "in" are discarded – only the vector in the final position will be decoded for generation. Thus the penultimate layer can be dramatically simplified: instead of $\frac{T^2+T}{2}$ attention pairs we only need to compute attention for the vector in the final position – resulting in precisely T attention pairs. All of the other attention pairs add nothing, as they are used to predict tokens which we already know. There appears to be a hint buried here: perhaps we could similarly discard the corresponding states in earlier layers? This would leave us only having to compute the residual tower sitting above the ultimate token.

We summarize our observations in Figure 3, showing a 3-layer transformer stack:

2.2. **InAttention Definition.** We recall a few of the observations we made in Section 2.1:

   1. Each residual tower allocates some (perhaps much) of its work and features to predicting the immediate next token, work which is irrelevant for residual towers to the right and invisible for those to the left.
   2. Work and features a residual tower "pays forward" to help predict future tokens is, by its nature, speculative and immediately depreciates in value when the actual next token is revealed.
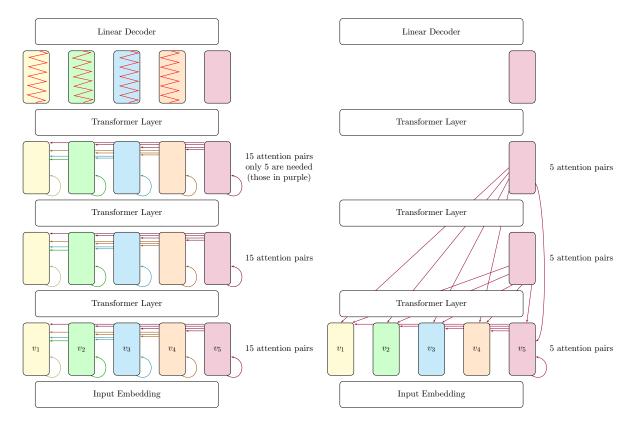
FIGURE 3. Left: Visualized attention pairs for dense attention. Right: Visualized attention pairs for inattention.

3. The penultimate layer of a transformer stack safely ignores all attention-pairs except those involving the final position. This means the final layer could safely have T (number of input tokens) attention pairs instead of the $\frac{T^2+T}{2}$ of other layers.

Given these observations, we recommend the following modification to the transformer stack: Have tokens attend initial states instead of states in their own row. During inference this means the vast majority of states need not be computed (only the ultimate residual tower), and the number of attention pairs goes down dramatically, and scales linearly. These changes are illustrated in Figure 3.

In some sense we are removing "self-attention" where columns in the tensor of hidden states attend each other, instead they attend the columns in the tensor of initial states. In another sense, however, this might be considered a variation of self-attention, where (a latent representation) of language tokens are attending (a latent representation) of themselves – we are merely changing where the latent representations are coming from.

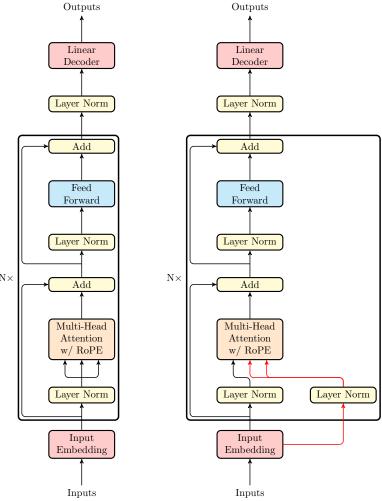Figure 4 [2] provides circuit diagrams for GPTNeoX models with and without our InAttention adjustment:

FIGURE 4. Left: GPTNeoX Model (without parallel residuals). Right: GPTNeoX Model modified to use InAttention. Red edges indicate the path of initial states.

Scaled dot-product attention (equation (1) from [16]) can be represented as

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

---

[2]Tikz code adapted from [10].

In a decoder-only transformer stack we have that $Q = W_Q X; K = W_K X; V = W_V X$ (assuming no biases). Here W_Q, W_K and W_V are linear operators and X represents the hidden-states of the model being fed in.

InAttention is very similar but instead $Q = W_Q X; K = W_K Y$ and $V = W_V Y$ where Y represents the initial-states of the model, which come straight from the Input Embedding and pass through a layer-specific Layer Norm.

3. BENCHMARKING INATTENTION

In this section we will quantify the effect that Inattention has on VRAM footprint and model capability. We will be training a series of baseline models using the $GPTNeoX$ architecture. We bring our own *Nazareth* tokenizer, which aims to be a word-level tokenizer specializing in English words. We refer to this series of baseline models as the $NazX$ series.

Modifying $GPTNeoX$ to implement InAttention is straightforward. We track the initial states and pass them to each layer. Attention queries are still computed by applying a linear transform W_Q to the hidden states, while the attention keys and values are now obtained by applying linear transforms W_K and W_V to the initial states. The only other architectural difference is that we initialize an additional layernorm for each attention block, which is applied to the initial states. This results in our $NazXN$, InAttention based model series having slightly more parameters than their $NazX$ counterparts.

Table 5 summarizes the hyper-parameters[3] of the $NazX$ and $NazXN$ model series:

Models				
Model Name	Embed Dim	MLP Factor	Layers	Parameters
$NazX235$	0768	3	12	235610880
$NazX420$	1024	4	16	421168128
$NazX735$	1280	5	20	733646080
$NazXN235$	0768	3	12	235629312
$NazXN420$	1024	4	16	421200896
$NazXN735$	1280	5	20	733697280

FIGURE 5. NazX(N) Family of Models

3.1. **VRAM Footprint.** The promise of Inattention is a substantially smaller VRAM footprint during inference. To illustrate this, we run queries of various lengths on $NazX420$ and $NazXN420$ architectures, keeping track of the VRAM usage and yielding the following results:

VRAM usage now scales linearly with context length and gains are substantial even for relatively short queries. Changing the model size will primarily affect the y-intercepts, so graphs would appear similar for various model sizes and architectures.

---

[3]All models use 8 attention heads.

VRAM Usage		
Query Length (Tokens)	NaxX VRAM (MB)	NazXN VRAM (MB)
Model Loaded	2945	2869
01024	4675	2965
02048	8271	2967
04096	18831	3231
08192	58339	4337
16384	OOM	6503
32768	OOM	10883

FIGURE 6. 420M Series VRAM Comparison

Notably, query lengths which cannot squeeze within an NVIDIA H100 graphics card for a typical transformer model (regardless of parameter count) fit comfortably on a 16GB VRAM GPU, not uncommon among consumers, using Inattention.
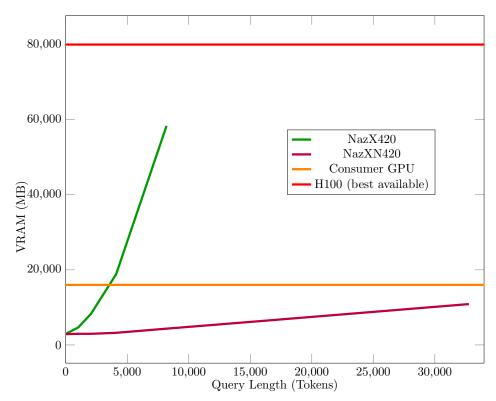

FIGURE 7. 420M Series VRAM Comparison

3.2. **Capability.** Of course running on a small amount of VRAM means nothing if Inattention-based models do not exhibit similar intelligence and scaling as their Attention-based counterparts, so we explore capabilities next.

In the following results, NazX and NazXN models were trained on the first $235, 420$ or 735 $C4$ files, respectively. We were unable to tokenize file 23 of $C4$, so it is omitted from all ranges and the subsequent file included instead (for instance, NazX420 was trained on files $[0, 22] \cup [24, 420]$).

Models were trained on a cluster of 16 $P4DE$ instances on AWS, each consisting of 8 A100 GPUs with 80GB VRAM. We used Deepspeed, leveraging the Hugging Face Trainer with default AdamW optimizer and cosine annealing scheduler with an initial learning rate of $2 * 10^{-4}$.

We adjusted batches per device for each model to roughly optimize VRAM usage but adjusted gradient accumulation steps in tandem so that the total number of forward steps (the product of batches per device, number of devices (which for us was $16 * 8 = 128$), and gradient accumulation steps) per optimizer step was $GPUS * BS * GAS = 12288$. The $235M$ and $420M$ models were trained with a batch size of 12, while the $735M$ models were trained with a batch size of 8.

We then run evaluations on file 1000 of $C4$, tokenized at different max context lengths, and report the average loss over this file in all upcoming figures.
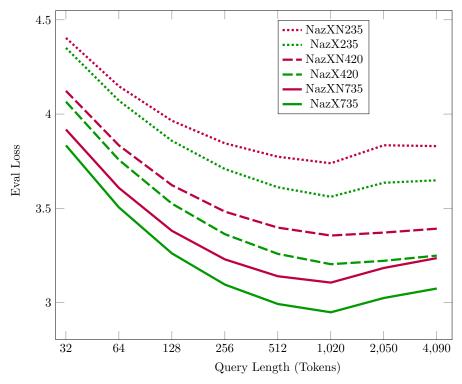

FIGURE 8. Loss By Context Length (Models Trained At CL=1024)

Capability degradation is non-negligible when switching to Inattention, but the tax does not appear to be growing with model size – we consider the trade quite favorable. The freed VRAM can be leveraged for capability gain in other ways – the most naive of which is to simply run a larger model.

Cutting the data another way, we below consider loss plotted against VRAM footprint for models running inference at a maximum context length of 1024 (which is what they were each pretrained with). While difficult to extrapolate from, the results for our six models are favorable towards InAttention and this advantage will rapidly increase as query lengths increase.
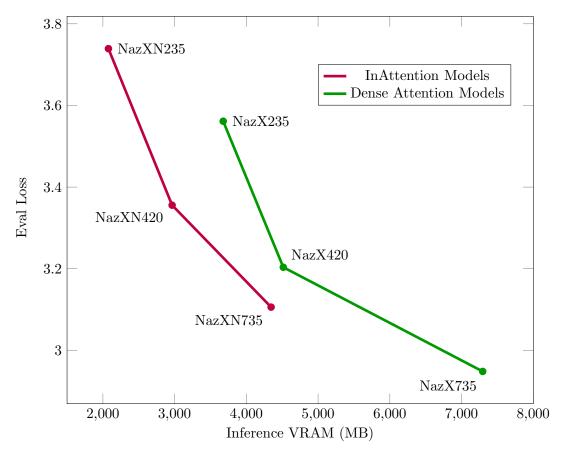

FIGURE 9. Loss By VRAM Footprint (At CL=1024)

4. Utilizing Long Context

Inattention allows substantially cheaper inference on long queries, but does not buy us any efficiency during training. Capability degradation of models past their trained context length is well established (e.g. [12], [14]), but we illustrate the problem by pretraining a NazX and NazXN model at a context length of 128, and then run inference at various
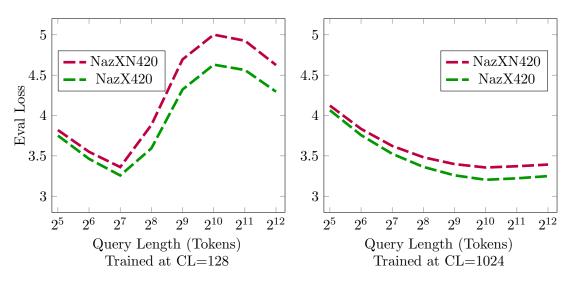
FIGURE 10. Length Extrapolation (Or Lack Thereof)

context lengths, noting substantial capbility degradation at unseen context lengths, see the left graph in Figure 10.

Models simply do not learn to extrapolate to unseen context lengths, despite our best attempts to provide positional encodings with built-in inductive biases.

We want our models to use as long of a context as possible, to better leverage the inference efficiency of Inattention, but the cost of doing so is prohibitive. A context length of 8192 does not fit on our GPUs if we train naively.

4.1. **Finetuning On Long Context Queries.** Luckily there is some evidence (e.g. [4], [15], [5]) that models can be cheaply fine-tuned to extend their context length. We seek to verify this independently:

Using a NazX420 model pretrained at context length 128 as a starting point we finetune on a single additional file of $C4$, file 421, tokenized at a context length of 1024. We treat this as a new training job with freshly initialized AdamW optimizer and cosine anneal scheduler with an initial learning rate of $2 * 10^{-5}$, one order of magnitude smaller than our initial pretraining learning rate. We also run an identical fine-tuning regime on a NazXN420 model pretrained at a context length of 128, presenting the results for both below.
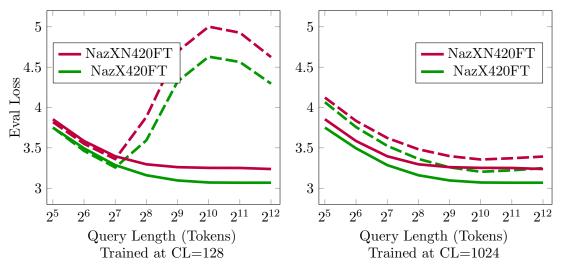
FIGURE 11. Finetuning to Extend Context Length

Indeed, finetuning on less than 1% of the initial training data allows the the CL128 models to dramatically improve their performance at CL1024, with only marginal degradation in performance on short context queries.

There are three things to observe in the figures above:

1. Finetuning on unseen context length data appears to be very effective at extending the context length a model knows how to use.
2. Models finetuned in this way better generalize to still unseen context lengths – note the slope is flat or slightly negative when increasing context length to the unseen lengths of 2^{11} and 2^{12}. This is in contrast to merely pretrained models which show performance degradation past their trained context length almost immediately.
3. The models pretrained at a context length of 128 and finetuned at 1024 outperform the models pretrained at context length 1024.

We urge caution when interpreting item (3) above – we suspect this is an artifact of our hyper-parameters: we did not increase the ratio of forward passes to optimizer steps for the models trained at a context length of 128, meaning those models received more optimizer steps than the models trained at 1024 context length (since the data was sharded into more, shorter batches).

The first two items, however, suggest a way forward: pretrain models at a reasonably long but still practical context length, then cheaply finetune them on much longer context lengths.

5. CONCLUSION AND NEXT STEPS

In this paper, we introduced InAttention, a modified attention mechanism that enables linear scaling of compute and memory with respect to context length during transformer inference. By having the hidden states at each layer attend to the initial token embeddings

rather than the previous layer's states, InAttention reduces the attention matrix to a vector and eliminates the need to cache intermediate activations.

Our experiments demonstrate that InAttention substantially reduces VRAM usage compared to standard dense attention, with the gains increasing for longer sequence lengths. This improved efficiency comes at the cost of model capability, as measured by evaluation loss, but this capability degradation can be accounted for by running larger models. We show that freed up memory from InAttention can be leveraged to deploy larger models that are more capable than smaller dense attention models with equivalent VRAM usage.

While InAttention provides a significant improvement in inference efficiency, it does not reduce the computational burden of training transformers with very long contexts. However, we corroborated recent findings showing that models can be cheaply finetuned to handle longer sequences than they were initially trained on, achieving strong performance. We believe the most promising path forward is to pretrain InAttention models with reasonably long contexts, then finetune them to fully exploit the inference benefits of InAttention on even longer sequences.

Future work should explore techniques to further optimize the efficiency of training InAttention-based models on long sequences. Combining InAttention with sparse attention mechanisms, more sophisticated finetuning methods, and other architectural innovations may lead to even greater gains. Overall, we believe InAttention is a valuable step towards practical and scalable transformer models that can fully leverage long-range context, and we are excited to see further developments in this direction.

References

[1] Mistral AI. Mistral transformer/sliding window attention, 2023.

[2] Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer, 2020.

[3] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

[4] Shouyuan Chen, Sherman Wong, Liangjian Chen, and Yuandong Tian. Extending context window of large language models via positional interpolation, 2023.

[5] Yukang Chen, Shengju Qian, Haotian Tang, Xin Lai, Zhijian Liu, Song Han, and Jiaya Jia. Longlora: Efficient fine-tuning of long-context large language models, 2024.

[6] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers, 2019.

[7] 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, 2022.

[8] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. Training compute-optimal large language models, 2022.

[9] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer, 2020.

[10] Miltos Kofinas. Tikz is all you need, 2022.

[11] Amirkeivan Mohtashami and Martin Jaggi. Landmark attention: Random-access infinite context length for transformers, 2023.

[12] Ofir Press, Noah A. Smith, and Mike Lewis. Train short, test long: Attention with linear biases enables input length extrapolation, 2022.

[13] Hongyu Ren, Hanjun Dai, Zihang Dai, Mengjiao Yang, Jure Leskovec, Dale Schuurmans, and Bo Dai. Combiner: Full attention transformer with sparse computation cost, 2021.

[14] Yutao Sun, Li Dong, Barun Patra, Shuming Ma, Shaohan Huang, Alon Benhaim, Vishrav Chaudhary, Xia Song, and Furu Wei. A length-extrapolatable transformer, 2022.

[15] Szymon Tworkowski, Konrad Staniszewski, Mikołaj Pacek, Yuhuai Wu, Henryk Michalewski, and Piotr Miłoś. Focused transformer: Contrastive training for context scaling, 2023.

[16] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[17] Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity, 2020.

[18] 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.