TRAIN SHORT, TEST LONG: ATTENTION WITH LINEAR BIASES ENABLES INPUT LENGTH EXTRAPOLATION

Ofir Press[1,2] **Noah A. Smith**[1,3] **Mike Lewis**[2]

[1]Paul G. Allen School of Computer Science & Engineering, University of Washington
[2]Facebook AI Research
[3]Allen Institute for AI
ofirp@cs.washington.edu

ABSTRACT

Since the introduction of the transformer model by Vaswani et al. (2017), a fundamental question remains open: how to achieve extrapolation at inference time to longer sequences than seen during training? We first show that extrapolation can be improved by changing the position representation method, though we find that existing proposals do not allow *efficient* extrapolation. We introduce a simple and efficient method, Attention with Linear Biases (ALiBi), that allows for extrapolation. ALiBi does not add positional embeddings to the word embeddings; instead, it biases the query-key attention scores with a term that is proportional to their distance. We show that this method allows training a 1.3 billion parameter model on input sequences of length 1024 that extrapolates to input sequences of length 2048, achieving the same perplexity as a sinusoidal position embedding model trained on inputs of length 2048, 11% faster and using 11% less memory. ALiBi's inductive bias towards recency allows it to outperform multiple strong position methods on the WikiText-103 benchmark. Finally, we provide analysis of ALiBi to understand why it leads to better performance.[1]

1 INTRODUCTION

When constructing a modern, large-scale, transformer-based language model, a major design decision is the length of training sequences (which up to now has been also equivalent to the length of inference sequences), denoted L throughout this paper. More context, achieved by larger L, enables better predictions at inference time. But longer sequences are more expensive to train.[2]

Before transformers, RNN language models were trained on shorter-L sequences and assumed to generalize to longer contexts at inference time (Mikolov et al., 2010; Mikolov & Zweig, 2012; Zaremba et al., 2014). Vaswani et al. (2017), introducing the transformer, speculated that it "may [...] extrapolate to sequence lengths longer than the ones encountered during training." We define *extrapolation* as the ability of the model to continue to perform well as the number of input tokens during validation increases beyond the number of tokens the model was trained on. We find that transformer LMs that use sinusoidal position embeddings have very weak extrapolation abilities; see Figure 1.

We demonstrate that this failure to extrapolate is caused by the position embedding method. As seen in Figure 1, recent alternatives to the original sinusoidal position method (Su et al., 2021; Raffel et al., 2020) have improved extrapolation. The better of these, the bias used in T5, is at least twice as slow as the sinusoidal approach and uses extra memory and parameters (Figure 2).

We introduce Attention with Linear Biases (ALiBi), which efficiently enables extrapolation. Compared to a sinusoidal model trained on the same input length, our method does not require additional

[1]We release our code and models at https://github.com/ofirpress/attention_with_linear_biases

[2]Figure 11 in the appendix plots training speed, in words per second, against L.

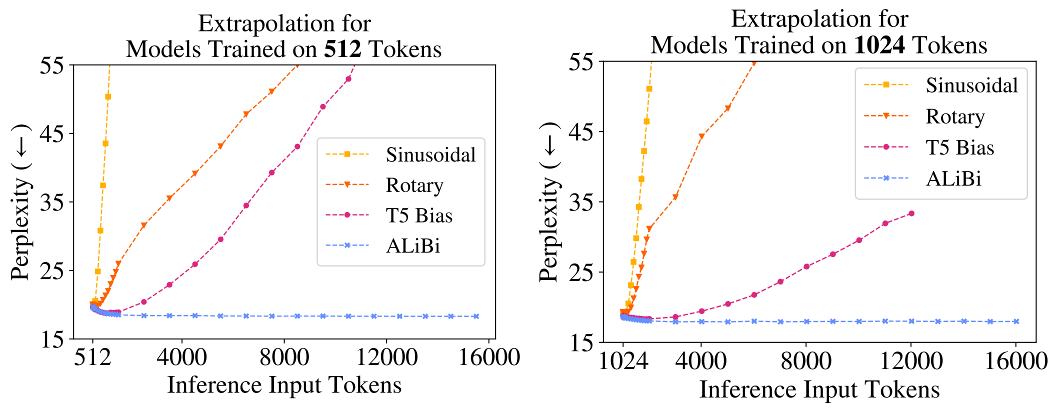

Figure 1: Extrapolation: as the (validation-set) input sequence gets longer (x-axis), existing position methods (sinusoidal, rotary, and T5) show degraded perplexity (y-axis, lower is better) while our method (§3) does not. Models trained on WikiText-103 with sequences of $L = 512$ (left) or $L =$ 1,024 (right) tokens. T5 ran out of memory on our 32GB GPU. For more details on exact perplexities and runtimes, see Tables 2 and 3 in the appendix.

runtime or parameters and incurs only a negligible (0–0.7%) memory increase. ALiBi can be implemented by changing only a few lines of existing transformer code.

Using ALiBi, a transformer LM can be trained on short-L sequences, and therefore at much lower cost, and still reliably applied to long texts at runtime. For example, a 1.3 billion parameter LM trained on $L = 1024$ tokens with ALiBi achieves the same perplexity as a sinusoidal model trained on $L = 2048$, when both are tested on 2048, even though our model is 11% faster and uses 11% less memory.

While performance peaks at around two times the number of tokens that model was trained on, ALiBi is still able to maintain strong performance even on sequences of length 10,000. In recently explored settings where NLP training examples are fed in as context to a LM (Brown et al., 2020), our approach will allow exposure to more examples. Additionally, it allows for generation of longer outputs.

We also investigate why ALiBi works so effectively. In §5, we find that ALiBi's improved performance is largely explained by its better avoidance of the early token curse. This suggests that future work building on ALiBi might achieve further gains by better exploiting longer histories.

2 Existing Approaches Don't Efficiently Extrapolate

We show for the first time that, while the sinusoidal position method technically should be able to extrapolate, in practice it has very limited extrapolation capabilities. We observe that the rotary position method improves over the sinusoidal method, but still does not achieve satisfying results. Keeping everything else constant, we then observe for the first time that the T5 bias position method extrapolates better than either of these, leading to the conclusion that extrapolation ability depends heavily on the position embedding. Unfortunately, this method is computationally costly (Figure 2).

2.1 Background and Experimental Setup

A transformer language model receives a list of tokens and outputs a probability distribution representing its prediction for the next token. We call the input list the *current input subsequence*, since typically the inputs to language models are subsequences from (much longer) training or evaluation sequences. During both training and perplexity evaluation (i.e., scoring a fixed sequence), many predictions can be calculated at once, using a "causal mask" that ensures each position's prediction is only influenced by tokens to its left. Let L be the length of each input subsequence during training; it includes L predictions which on average have access to $\frac{L+1}{2}$ tokens of (left) context. Increasing

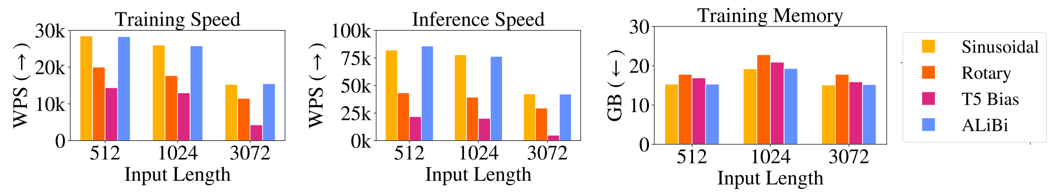

Figure 2: A comparison of batched training and inference speed and memory use of the sinusoidal, rotary, T5 bias, and our ALiBi position methods. The speed differences between our method and the sinusoidal are within 1% during training and 3% for inference, which is insignificant on our hardware. ALiBi uses 100MB extra memory when training on input lengths 1024 and 3072 in this setting. Note that memory usage is lower in all approaches, when training on 3072 tokens (compared to 1024) since when training on 3072 tokens we break batches into multiple updates. See Table 1 in the appendix for the exact numbers.

L increases the amount of context available (to the average prediction) during training and evaluation. To explore the extrapolation abilities of a model, we are interested in cases where sequences of length $L_{valid} > L$ are considered at evaluation time. In cases where L is different during inference than what it was during training, we use L to refer to the length of subsequences used during training and we use L_{valid} to refer to the length of subsequences used at validation.

Nonoverlapping Inference To train on or evaluate a sequence longer than L tokens, it is typical to segment the sequence into L-length subsequences and train or evaluate them independently. Unless otherwise stated, we use nonoverlapping inference in reporting perplexity scores.

Extrapolation during Inference Formally, the functions that define a transformer layer are agnostic about the input length;[3] they map from some arbitrary, unfixed number of input vectors to the same number of output vectors. When transformers are applied to data that is inherently sequential, like text, positional information is injected into the inputs in various ways.

Vaswani et al. (2017) discussed two options for embedding positions into vectors to be added to word embeddings: learning embeddings for specific positions and unlearned sinusoidal embeddings. They observed similar performance between these two, but preferred the sinusoidal approach, which they argued might extrapolate to longer input sequences during inference. We find that this model can not extrapolate to more than a few dozen tokens beyond L.[4]

Experimental Setup We first test the extrapolation abilities of various position methods on the WikiText-103 corpus (Merity et al., 2016) using the transformer language model of Baevski & Auli (2018). We use this model because of its prominent role in recent language modeling developments (Khandelwal et al., 2020; Press et al., 2021). The training set is about 103 million tokens from English Wikipedia (half a gigabyte). The model has 16 transformer layers of dimension 1024, with 8 heads, and the feedforward inner dimension is of 4096. This model ties the word embedding and softmax matrices (Press & Wolf, 2017; Inan et al., 2017). In our experiments, other than varying the position method and training subsequence length, we modify no other hyperparameters, including the random seed and number of training epochs (205).

2.2 Sinusoidal Position Embeddings

Sinusoidal position embeddings (Vaswani et al. (2017); §3.5) are constant non-learned vectors that are added to token embeddings on input to the first layer of the transformer. They are frequently

[3]These include the embedding lookup, feedforward sublayer, and softmax layer that act independently on the vector inputs, and the attention sublayers whose parameters do not depend on the input length (and which must handle variable-length inputs, e.g., due to causal masking). We omit the details of the transformer architecture because they are orthogonal to our exploration.

[4]The learned positional embedding approach does not have a way to encode positions greater than L; it therefore has no hope of extrapolation.

used in transformer language modeling (Baevski & Auli, 2018; Lewis et al., 2021), and machine translation (Vaswani et al., 2017; Ott et al., 2018) models. First, we take the unmodified model of Baevski & Auli (2018), which uses sinusoidal position embeddings, and after training it on $L = 512$ tokens, we run inference with it on the validation set on $L + k$ tokens, for k ranging from 0 to 15,000. Figure 1 (left) and the corresponding Table 2 (in the appendix) show that while the model is able to improve perplexity up to $k = 20$, performance stops improving and stays steady from $k = 20$ to $k = 50$, and then starts degrading. Similar results are obtained for a model trained with $L = 1024$ tokens (Figure 1 right, and Table 3 in the appendix). That model is able to improve for up to $L_{valid} = L + 50$ tokens and then performance starts degrading.

2.3 Rotary Position Embeddings

The rotary position method was introduced by Su et al. (2021) and has recently been popularized by the open source GPT-3 (Brown et al., 2020) implementation GPT-J (Wang & Komatsuzaki, 2021). Instead of adding sinusoidal embeddings at the bottom of the transformer network, they multiply the keys and queries of every attention layer by sinusoidal embeddings.

Note that, as opposed to the sinusoidal or learned positional embedding approach, the rotary method injects position information into the model at every layer, and not just at the initial one. In addition, it does not add any position information to the values of the self-attention sublayer. The output of a self-attention sublayer is a linearly-transformed, weighted sum of the input value vectors, and so by not inserting position information into the values, the outputs of each transformer-layer do not have any explicit position information in them. We suspect that this segregation of position information may be beneficial for extrapolation, and we draw inspiration from it in the design of our method (§3).

We apply the rotary position embedding method to our Baevski & Auli baseline.[5] The perplexity results (Figure 1, and Tables 2 and 3 in the appendix) are better than the sinusoidal approach as the model with $L = 512$ ($L = 1024$) is able to improve perplexity with up to $k = 200$ ($k = 100$) additional tokens than it saw during training, but this comes at the cost of slower training and inference (Figure 2).

2.4 T5 Bias

While most models use trained or sinusoidal position embeddings, the T5 model of Raffel et al. (2020) uses a relative position method (Shaw et al., 2018; Huang et al., 2019) that does not add any kind of position information to the word embeddings (as in the previous method) , but instead modifies the way attention values are computed. We refer to this method as the “T5 bias.” In the unmodified transformer, to compute attention values we compute the dot product of every query with every relevant key and then softmax these raw attention values. In the T5 bias method, we compute the attention values as before but then we add a learned, shared, bias to each query-key score, that is dependent on just the distance between the query and key. So all query-key scores where the query and key distance is zero (so the query is attending to itself) get a specific bias, all scores where the query and key are one word away get a different bias, and so on, up to a certain point, from where multiple different distances share the same bias (which might be beneficial for extrapolation). Note that as in the rotary method, the T5 bias injects position information into the model at every layer, and does not integrate any explicit position information into the self-attention value vectors.

Raffel et al. (2020) propose that the T5 bias may allow extrapolation, but they did not report experiments testing this property. Here we show that the T5 bias method does indeed allow language models to extrapolate. We do this by again modifying the Baevski & Auli model, this time to insert the T5 bias into it.[6]

[5]Our RoFormer implementation is based on the code from `https://github.com/JunnYu/RoFormer_pytorch` which is linked to from the official RoFormer repository (`https://github.com/ZhuiyiTechnology/roformer`). After we finished running our experiments with the rotary method we were made aware that the runtime of the code linked to above could be optimized, leading it to be just 2% slower than the sinusoidal approach. Applying this optimization would not lead to a change in extrapolation performance.

[6]Our T5 bias implementation is based on the T5 implementation in HuggingFace Transformers (Wolf et al., 2020) which itself is based on the official Mesh Tensorflow T5 code.

As Figure 1 shows, the T5 bias is able to improve perplexity with longer sequences than the ones it was trained on by $k = 600$ ($k = 800$) extra tokens for a model trained on $L = 512$ ($L = 1024$) input tokens. Unfortunately, this impressive performance comes at a cost: batched training is at least twice as slow as the sinusoidal model, therefore this model's extrapolation ability does not actually help in practice. For example, if we wanted to do inference on 1024 tokens, we could either train the sinusoidal model with L = 1024 or train the T5 bias model on L = 512 tokens and extrapolate to 1024 for inference. But the L = 1024 sinusoidal model runs at 28.5k words per second (WPS), while the L = 512 T5 bias model runs at 14.4k WPS (Appendix Table 1), so there isn't actually a benefit to training on shorter sequences with the T5 bias method.

3 Attention with Linear Biases (ALiBi)

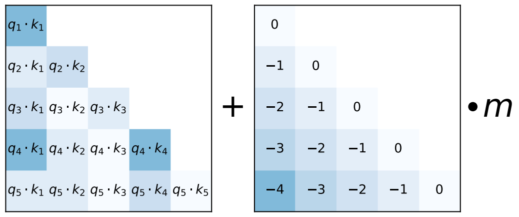
$q_1 \cdot k_1$
$q_2 \cdot k_1$ $q_2 \cdot k_2$
$q_3 \cdot k_1$ $q_3 \cdot k_2$ $q_3 \cdot k_3$
$q_4 \cdot k_1$ $q_4 \cdot k_2$ $q_4 \cdot k_3$ $q_4 \cdot k_4$
$q_5 \cdot k_1$ $q_5 \cdot k_2$ $q_5 \cdot k_3$ $q_5 \cdot k_4$ $q_5 \cdot k_5$
+
0
−1 0
−2 −1 0
−3 −2 −1 0
−4 −3 −2 −1 0
• m

Figure 3: When computing the attention scores for each head, our linearly biased attention (ALiBi) method adds a constant bias (right) to each attention score ($\mathbf{q}_i \cdot \mathbf{k}_j$, left). As in the unmodified attention sublayer, the softmax function is then applied to these scores, and the rest of the computation is left unmodified. m is a head-specific scalar that is set and not learned throughout training. We show that our method for setting the m values generalizes to multiple text domains, models and training compute budgets. When using ALiBi we do *not* add positional embeddings at the bottom of the network.

In the transformer model of Vaswani et al. (2017), position embeddings are added to the word embeddings at the bottom of the network. For an input subsequence of length L, the attention sublayer applies the following computation to compute the attention scores for the ith query $\mathbf{q}_i \in \mathbb{R}^{1\times d}$ ($1 \leq i \leq L$) in each head, given the first i keys $\mathbf{K} \in \mathbb{R}^{i\times d}$, where d is the head dimension:

$$\text{softmax}(\mathbf{q}_i\mathbf{K}^\top)$$

These attention scores are then multiplied by the values to return the output of the attention sublayer.[7]

When using ALiBi we do not add position embeddings at any point in the network. The only modification we apply is that after the query-key dot product we add a static, non-learned bias:

$$\text{softmax}(\mathbf{q}_i\mathbf{K}^\top + m \cdot [0, -1, -2, ..., -(i-1)])$$

Where scalar m is a head-specific slope fixed before training. Figure 3 offers a visualization.

For our models that have 8 heads the slopes that we used are the geometric sequence that starts at $\frac{1}{2}$ and multiplies each element by $\frac{1}{2}$ to compute the next element: $\frac{1}{2^1}, \frac{1}{2^2}, ..., \frac{1}{2^8}$. For models that require 16 heads we interpolate those 8 slopes by geometrically averaging every consecutive pair, resulting in the geometric sequence that starts at $\frac{1}{\sqrt{2}}$ and has the ratio of $\frac{1}{\sqrt{2}}$: $\frac{1}{2^{0.5}}, \frac{1}{2^1}, \frac{1}{2^{1.5}}, ..., \frac{1}{2^8}$. In general, for n heads, our set of slopes is the geometric sequence that starts at $2^{-2^{-(\log_2 n)-3}}$ and uses that same value as its ratio.

In §4, we observe that this set of slopes works on a wide variety of text domains and model sizes, and so we do not believe that these slope values should be tuned every time a new model is trained on a new dataset. This makes our method similar to the sinusoidal approach, where the hyperparameters (the start and end of the geometric progression of wavelengths) were set once by Vaswani et al. (2017) and then reused in different models of different sizes on different datasets.

[7]For simplicity, we have omitted the key, query and value matrix multiplication, the final output projection and the dropout module.

ALiBi has an inductive bias towards recency, by penalizing attention scores between distant query-key pairs, with the penalty increasing as the distance between a key and a query grows. The different heads increase their penalties at different rates, depending on the slope magnitude.

We initially experimented with making the slopes trainable, but this did not lead to strong extrapolation results.[8] A brief manual exploration of around ten slope sets led us to discover the set of slopes that we finally picked. Our main insight from this exploration is that the slope sets that work best are ones in which the slopes are in the $(0, 1)$ range, with the density of the slopes increasing as we get closer to 0. During that exploration we found our method to be robust to slope choice. Even randomly sampling from the exponential distribution could work well in some cases (although that method had high variance).

Note that our ALiBi method is a relative position method, and like both the T5 bias and the rotary method, we add position information at every layer to the keys and queries but not to the values. We hypothesize that these properties might be beneficial for extrapolation.

Implementation ALiBi is easy to implement, with all of the changes accomplished in a few lines of code. We implement it by adding the linear biases to the mask matrix. In practice, when training a transformer language model, query $\mathbf{q}_i$ attends only to keys 1 to i, and that is implemented by adding a mask matrix to the query-key dot product before the softmax operation is applied. This means that there is no runtime penalty when using our method, since we do not add any operations to the network, we only modify the mask that is added to the attention scores in every attention sublayer.

Compared to the sinusoidal model trained on the same input lengths, there is a memory increase (up to 100MB in some of our experiments) when using ALiBi since in the unmodified transformer the mask is of size $L \times L$, but when using ALiBi the mask is slightly larger, at $n \times L \times L$ (where n is the number of heads) since the linear biases added for each head must use a different slope. But, as we show, ALiBi allows for training on much smaller sequences while still achieving (and sometimes surpassing) results obtained using sinusoidal embeddings on longer sequences, and this saves multiple gigabytes of memory.

4 Results

We first present results on WikiText-103, showing that our ALiBi method is efficient and allows for training models with short input subsequences that outperform strong baselines even when extrapolating to more than six times the number of tokens that they were trained on. We then take the same hyperparameters for our method (the set of slopes) that worked on WikiText-103 and show that without any modification they provide strong results on a dataset in a very different domain than WikiText-103: books. Finally, we show that a 1.3B parameter model trained on a much larger (461 GB) dataset, using much more compute, with our ALiBi method provides a superior alternative to the sinusoidal method, as it achieves similar perplexity scores while using significantly less memory.

While multiple alternatives to the position methods presented in Vaswani et al. (2017) have been proposed, few have been adopted in large (1B or more parameters) language models, as that setting is much more challenging than the smaller scale experiments. GPT-3 and Jurassic-1 (Lieber et al., 2021) use the learned position embedding method from Vaswani et al., and GPT-J uses the rotary method. Our results on the 1.3B parameter model show our method's ability to generalize to greater models, dataset size and training duration, without retuning the hyperparameters.

4.1 Results on WikiText-103

We first develop our method on the WikiText-103 corpus (Merity et al., 2016), replacing the sinusoidal position embeddings in the language model of Baevski & Auli (2018) with ALiBi.

Figure 4 (and the corresponding Table 5 in the appendix) show our results, for models trained with varying numbers of input subsequence tokens (L), extrapolating to longer subsequence lengths on the validation dataset. The first observation is that, without extrapolation, for every L, our models beat the ones that use the sinusoidal method, sometimes by a significant amount. For example, the

[8]Another issue with trainable slopes is that it slows down training speed by 3%.

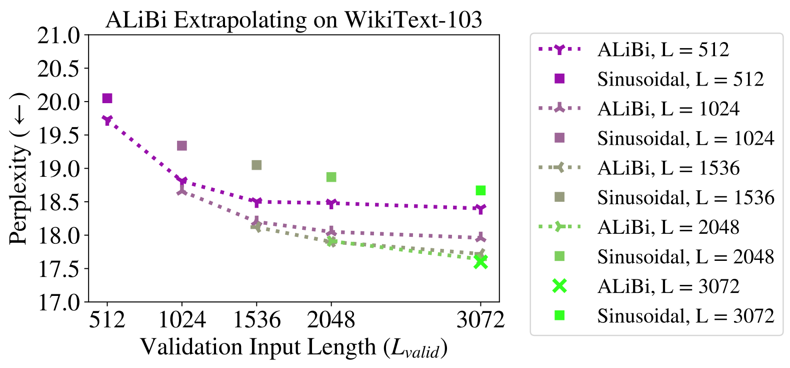

Figure 4: Our ALiBi-enabled models trained and evaluated on different sequence lengths, on the WikiText-103 validation set, compared to the sinusoidal baseline (not evaluated on longer input sequences). All of our models manage to beat the sinusoidal baseline, even when trained on less tokens. See appendix Table 5 for the exact perplexities, more ALiBi models (trained on fewer tokens), and the results of the rotary and T5 bias models.

Baevski & Auli model achieves 18.67±0.24 (std. dev.) perplexity when trained with $L = 3072$ input tokens, but our $L = 3072$ model achieves 17.60 perplexity (when both models evaluate with L_{valid} = 3072).

The second observation is that all of our models are able to extrapolate and they manage to obtain improved perplexity scores when handling more tokens than they observed during training. For example, our model trained on 512 tokens (which achieves 19.73 perplexity when evaluating subsequences of length 512 in the development set) is able to achieve a perplexity score of 18.40 on the development set when extrapolating to subsequences of length 3072. Surprisingly, this beats the score that the $L = 3072$ sinusoidal model obtains on the development set, by a statistically significant margin. Note that all our models trained on $L = 512$ to $L = 2048$ are able to beat the sinusoidal baseline trained on $L = 3072$ when extrapolating to L_{valid} = 3072, even though all of those models take much less time to train since they train on shorter subsequences! The $L = 512$ model is 1.84 times faster to train and yet it still beats the $L = 3072$ when extrapolating to $L_{valid} = 3072$. In addition, training the $L = 3072$ sinusoidal model requires a GPU with more than 16 GB of memory to fit the large attention matrices, but our $L = 512$ beats its performance even though it can be trained on a GPU with much less memory, due to the much smaller attention matrices.

Additionally, Table 5 (in the appendix) also shows that, for L of 1024 and 3072, our method performs better than the rotary and T5 bias models, even when L_{valid} = L. Figure 1 (and the corresponding Appendix Tables 2 and 3) contain a wider exploration of our method against the other position methods. They show that while the T5 bias (the best of the baselines) is able to improve perplexity until L_{valid} is around $2L$, on the WikiText-103 dataset our method continually improves perplexity until at least around $3L$, with the $L = 512$ model being able to improve perplexity even when L_{valid} is more than 12k tokens. Even when not being able to improve perplexity given longer sequences, ALiBi always maintains strong performance as more tokens are added.

Figure 5 is a cross section of Figure 4, showing our models with different train lengths and the sinusoidal baseline, all evaluated on L_{valid} = 3072 tokens. It shows that all of our models with $512 \leq L < 3072$ are faster to train than the sinusoidal model with L = 3072, but they all beat its perplexity score on the validation set. Our model with L = 3072 trains just as fast as the sinusoidal one but beats its score by more than one perplexity point (the standard deviation for the the sinusoidal model with L = 3072 is 0.24).

Appendix Table 6 shows that our results on the validation set also transfer to the test set of WikiText-103.

Currently, almost all other models that present results on WikiText-103 use sliding window evaluation (defined in §5) to compute perplexities. We apply that method to our models (and to our sinusoidal, rotary and T5 bias baselines) in Appendix Table 7 and find that our L = 3072 model

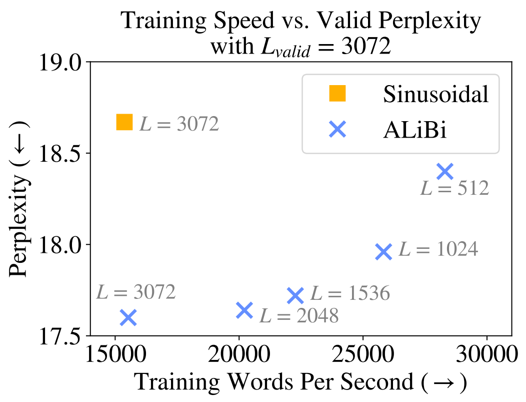

Figure 5: The training speed and validation perplexity (with $L_{valid} = 3072$) for ALiBi models and the sinusoidal model trained with $L = 3072$. All of our models trained on 512 or more tokens achieve better perplexity than the sinusoidal model, even though all of them (except the $L = 3072$) require less time and memory to train.

surpasses the performance of Transformer-XL (Dai et al., 2019), the Sandwich (Press et al., 2020) and Shortformer (Press et al., 2021) models. Our results are similar to the ones obtained with staged training (Press et al., 2021), but are behind the results obtained by the Routing Transformer Roy et al. (2020) and the kNN-LM Khandelwal et al. (2020). The methods used in those models are orthogonal to ours and we hypothesize that combining them with our might lead to an even bigger performance increase.

4.2 Results on the Toronto Book Corpus

To ensure that our results aren't particular to the WikiText-103 corpus, we next apply our model and the baselines to a different domain, while using a similar model architecture and the same ALiBi slopes as the ones used in the previous subsection.

We emphasize that our set of slopes was chosen by running experiments on the WikiText-103 corpus, and in this section we will take that set of slopes that we picked and apply it to a model trained on a very different text domain. Throughout the entire process of developing this method, we only ran one set of experiments on this domain, using the set of slopes we previously picked.

Specifically, we use the Toronto BooksCorpus (Zhu et al., 2015), which has previously been used to train BERT (Devlin et al., 2019), in addition to English Wikipedia. The corpus is about 700M tokens (2.9 GB).

We use the same train/validation/test split as Khandelwal et al. (2020), as well as their tokenization, which uses BERT's vocabulary of 29K byte-pair encodings. Since the vocabulary is much smaller than WikiText-103's, we replace the adaptive word embedding and softmax of Baevski & Auli (2018) with a tied word embedding and softmax matrix (Press & Wolf, 2017; Inan et al., 2017).

Our results in Figure 6 (and Table 8 in the appendix) replicate our success on the WikiText-103 dataset. Our model is able to beat the sinusoidal baseline when trained on the same amount of input tokens (L) and in addition, our model is able to extrapolate to longer sequences at inference. This occurs even though our set of slopes was *not* tuned on this dataset. This shows the generality of ALiBi and the particular set of slopes we found, and hints that they may be used on different text domains without further hyperparameter tuning.

Appendix Tables 9 and 10 present the perplexities for our ALiBi models, the baselines, and the current state of the art on the Toronto BookCorpus validation and test sets. Our results here mirror our results on WikiText-103: we improve over the sinusoidal baseline, even when trained on less tokens.

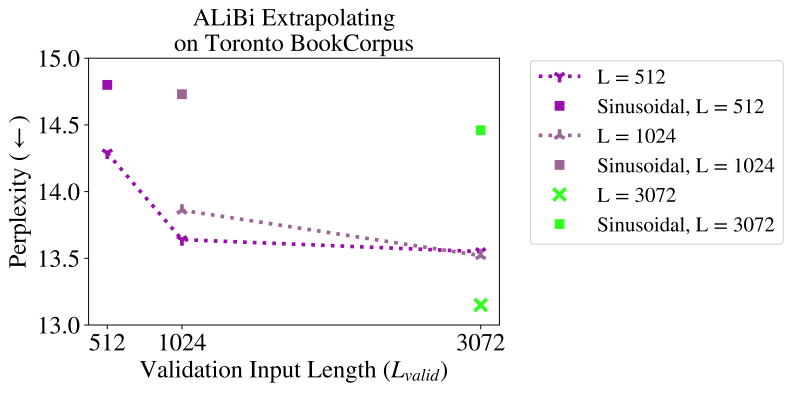

Figure 6: ALiBi-enabled models evaluated on different input lengths, on the Toronto BookCorpus. Our models extrapolate to longer sequence lengths and are able to beat the sinusoidal baseline, even when trained on much shorter sequences.

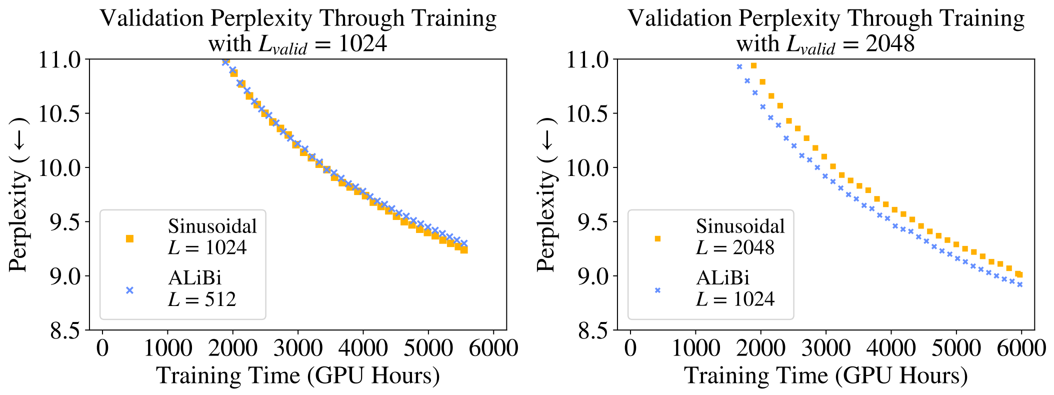

Figure 7: On the left (right), ALiBi-enabled models trained on 512 (1024) and evaluated on 1024 (2048) tokens, during training, compared to the sinusoidal baseline trained on 1024 (2048) tokens. ALiBi obtains strong results even though our models here use 6%-11% less memory, since they train on shorter sequences. See Appendix Table 11 for the precise memory use values and end-of-training perplexities.

4.3 RESULTS ON THE CC100+ROBERTA CORPUS

Our final set of experiments investigates whether ALiBi transfers to a bigger model trained with a bigger computational budget on a bigger dataset than the ones we have previously trained on. We show that our method achieves strong results in this more challenging setting, obtaining similar performance to the sinusoidal baseline while using significantly less memory, since we train on shorter subsequences.

The dataset we choose is a combination of the datasets used to train the RoBERTa (Liu et al., 2019) implementation of BERT (Devlin et al., 2019) and the English part of the CC-100 corpus introduced in Conneau et al. (2020) for a total of 461 GB. The RoBERTa training corpus is 161 gigabytes—the Toronto Book Corpus (Zhu et al., 2015), English Wikipedia, CC-News (Nagel, 2016), OpenWebText (Gokaslan & Cohen, 2019) and Stories (Trinh & Le, 2018))—and the English part of the CC-100 corpus is 300 gigabytes. The validation set contains 649K tokens.

Our models for this dataset have 25 transformer layers with 16 heads and a dimension of 2048 with the hidden dimension of the feedforward sublayers being 8192. These models have 1.3B parameters. We train our models for one epoch, which is 50k updates on 128 V100 GPUs.

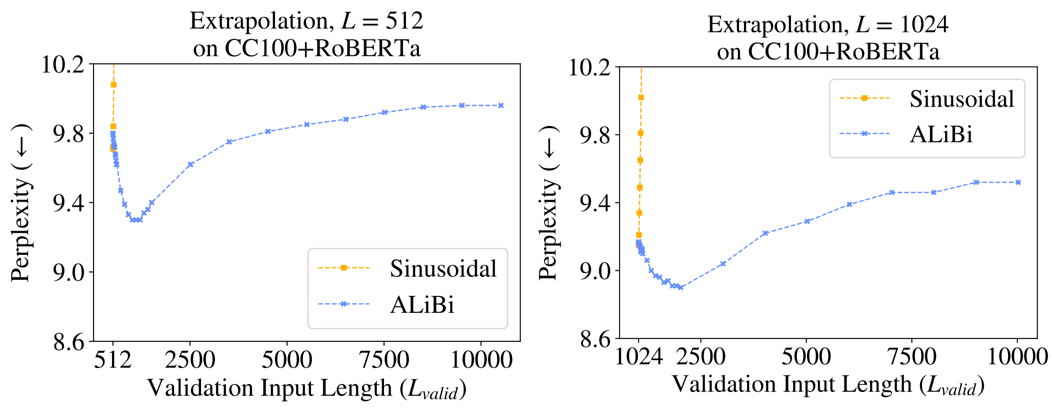

Figure 8: Our model and the sinusoidal baseline (with both L = 512 and 1024), all trained for 50k updates (1 epoch) on the CC100+RoBERTa corpus, extrapolating on the validation set. ALiBi achieves the best results at around $2L$, but maintains strong performance up to 10000 tokens in these experiments.

In Figure 7 (left) we compare the validation perplexity, for L_{valid} = 1024, throughout the training process for a model trained with our ALiBi method on L = 512 compared to the sinusoidal baseline trained with L = 1024. Since our model is trained on much shorter sequences, it is 7% faster and uses 1.6 GB less memory. We halt training of the sinusoidal baseline when our model reaches the end of its training (one epoch). At the end of training, our model is just 0.06 perplexity away from the baseline, even though it was trained on sequences that are half the length of the ones the baseline was trained on, and used less memory.

In Figure 7 (right), the results are even better, showing that our model trained on L = 1024 is able to beat by 0.09 perplexity the sinusoidal model trained on L = 2048 (when evaluating with L_{valid} = 2048), even though our model uses 3.1 GB less memory. As Figure 7 (right), it maintains a lead in perplexity over the sinusoidal model during the entire training process. By sampling five evenly distributed points across the training process, we compute that our L = 1024 model reaches a given perplexity value, on average, 11% faster than the sinusoidal model does.

Since our models in these comparisons use much less memory, they allow for stacking more layers which would further improve performance (with negligible, if any, runtime cost). To keep our experiments as straightforward as possible, we do not add layers to our models.

Appendix Table 12 has additional results comparing our models to the sinusoidal baseline when both are trained on the same L, showing that ALiBi performs similarly to the sinusoidal baseline when not extrapolating. This is in contrast to the results presented on the smaller datasets, where ALiBi consistently outperformed other position methods, even when not extrapolating. This hints that ALiBi's inductive bias is provides additional benefits for lower-resource language modeling.

Figure 8 shows that our models trained on $L = 512$ and $L = 1024$ are able to achieve the best results when extrapolating to about double the tokens that they were trained on. Specifically, the L = 512 (that obtains 9.79 perplexity when L_{valid} = 512) achieves its best score (9.3) when extrapolating to 1012 tokens, and the L = 1024 (that obtains 9.16 perplexity when L_{valid} = 1024) achieves its best score (8.9) when extrapolating to 2024 tokens.

One possible explanation is that during training the subsequences the model observes are up to L tokens long. When performing inference on subsequences of length $2L$, half of the subsequences the model consumes are as long as the examples seen during training. When inference is performed on subsequences of length $2L + 1$ or longer, less than half of the predictions the model makes are on subsequences of lengths seen during training, and that might hurt performance.

Note that the sinusoidal model cannot extrapolate at all in this setting, with performance degrading for both the L = 512 and 1024 models as soon as one token more than L is added during evaluation.

5 Analysis

5.1 Defining sliding window evaluation and the early token curse

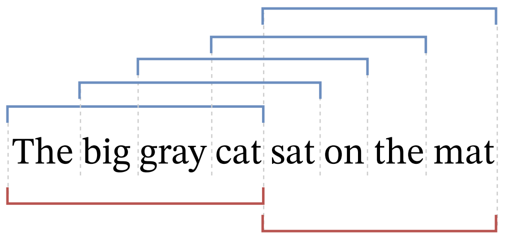

Figure 9: Sliding window evaluation (top; blue) compared to nonoverlapping evaluation (bottom; red) on a sequence of 8 words using a model with L_{valid} = 4. Nonoverlapping evaluation is much faster since it requires just two inference passes (as opposed to the five passes required by the siding window approach). But the sliding window approach provides more context for each prediction.

Sliding Window Inference As mentioned in Section 2, nonoverlapping inference is commonly used to evaluate sequences longer than L (number of tokens in each training subsequence). An alternative to the that is to use a sliding window during evaluation (Baevski & Auli, 2018).

A stride S is picked between 1 and $L - 1$ and advance the window by S tokens after each forward pass.[9] This means that $L - S$ tokens from the previous subsequence are re-encoded, and only S new tokens are outputted. The advantage is that all outputs in each subsequence after the first have at least $L - S$ previous tokens to condition on. However, since tokens must be re-encoded multiple times, this approach is much slower than the nonoverlapping one. When $S = 1$, we output one token every inference pass, each using the maximal context window that the model can handle, but this is the slowest approach. Figure 9 is a visualization of the nonoverlapping and sliding window evaluation approaches.

We use sliding window inference as a tool to analyze our models, but note that it is normally prohibitively slow in practice (Press et al., 2021).

Early Token Curse Splitting an evaluation set into subsequences means that predictions that occur early on in each subsequence will not have access to many previous context tokens (which appeared at the end of the previous subsequence). The result, referred to as the *early token curse* (Press et al., 2021), is an increase (i.e., degradation) to perplexity scores. A workaround is to evaluate the model using a sliding window, giving each prediction more context. This solution is slow since it requires many more forward passes of the model.

5.2 Extrapolation reduces the early token curse

We have presented results showing that our ALiBi method (and, to a lesser extent, the T5 bias) allow language models to extrapolate during inference. There could be two reasons for these models achieving better perplexity given longer input subsequences:

1. The model performs better because it is able to use the longer contexts to make more accurate predictions. For example, the average article length in the WikiText-103 corpus is about 3600 tokens, and so if a model trained on $L = 512$ tokens extrapolates to L_{valid} = 3072 tokens during inference and achieves better results, that might be because the model is able to spot patterns occurring across more than 512 tokens.
2. The model performs better because longer input sequences provide more context and so the early token curse is reduced. When we evaluate a validation set with a model with some input subsequence length L_{valid} , we're actually making L_{valid} predictions, where the

[9]Nonoverlapping inference can be viewed as sliding window inference with stride L.

first prediction has just one token of context, the second has two tokens, and so on. So the average prediction gets to use $\frac{L_{valid}+1}{2}$ tokens of previous context. When we extrapolate and increase L_{valid} , the amount of tokens of context we have in the average prediction also increases. And so the performance improvement might not come from being able to do better on the longer sequences, but because our average number of context tokens has become larger for each prediction.

To better understand what might be occurring, we re-evaluate the development set of WikiText-103 with our models (and the sinusoidal baseline) with $L = 512, 1024, 3072$, but this time we use sliding window evaluation, with a stride of $S = 1$, meaning that we move the sliding window just one token after every inference pass, making each prediction with the maximal number of tokens that that model can use.

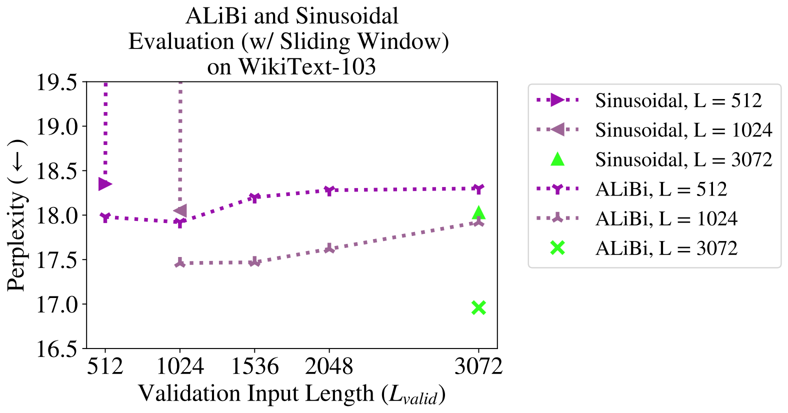

Figure 10: ALiBi models evaluated on different input lengths, on WikiText-103 with sliding window evaluation (with stride $S = 1$). Unlike in Figure 4, where performance improves in each of our models as we increase the validation sequence length, here performance stays relatively flat as we increase L_{valid} . This might mean that ALiBi increases performance when $L_{valid} > L$ not because it is able to use longer contexts, but because fewer tokens will suffer from the early token curse. Note that as in §2, the perplexity of the sinusoidal model explodes when $L_{valid} > L$, even when using sliding window evaluation.

The results are shown in Figure 10 and in the corresponding Appendix Tables 13 (sinusoidal) and 14 (ALiBi).

For the sinusoidal model, as in §2, increasing L_{valid} leads to an explosion in perplexity, even when using sliding window evaluation. Our ALiBi models are not able to improve perplexity when looking at longer sequences in this setting, but manage to keep perplexity flat when L_{valid} is increased.

This leads us to believe that our perplexity improvement when increasing L_{valid} and using nonoverlapping evaluation is caused by explanation 2, and not by explanation 1. Our ALiBi results mirror what occurs in the model that uses the T5 bias: when using sliding window evaluation, perplexity remains relatively flat when evaluating longer subsequences (see appendix Table 15).

This analysis reveals that when $L_{valid} > L$ ALiBi might not be using contexts longer than the ones it was trained on, and this points to a research direction which could be pursued in future work.

These findings do not lessen the value of ALiBi. When $L_{valid} = L$, ALiBi achieves either superior or similar results to the sinusoidal method and other alternatives, even though it is simpler and does not require any learned parameters. When evaluating $L_{valid} > L$ tokens, even if it turns out that ALiBi doesn't attend to more than L tokens, it leads to better results than the other alternatives that can be used in this case, with standard nonoverlapping inference (which is cheap, but doesn't perform as well) and with the more accurate sliding window approach (which is very slow).

6 Related Work

In parallel with our work, Wennberg & Henter (2021) introduce a relative position embedding that, like our method, adds a bias to the attention scores that is a function of the distance between the key and query elements. Unlike our ALiBi method, which uses a non-learned linear function, their method uses a radial-basis function, with multiple trainable parameters (which in our experiments led to a slight decrease in runtime). In addition, they present experiments on text classification, and not on language modeling. They do not explore extrapolation.

Recently, the Distance Aware Transformer (Wu et al., 2021) was introduced. This transformer multiplies attention scores by a bias that is a function of the distance between the key and query elements. This function uses a different, learned, parameter in every head. They only show results on text classification. In our experiments (not presented), multiplying the attention scores by the bias (instead of adding, as in ALiBi) led to degraded performance.

Transformer-XL (Dai et al., 2019) presented a language model that uses a cache and can attend to more tokens during inference than it was trained on (by increasing the length of the cache), but they only present results where the output length is limited to the L (the training length), and their relative position method is very slow (Press et al., 2021).

The Longformer (Beltagy et al., 2020) presented a method for adapting models trained on shorter sequences to document-level tasks, but in order to achieve this they had to partially train their models on longer sequences. Our ALiBi method enables extrapolation without any additional training on longer sequences.

7 Conclusion

We showed that the popular sinusoidal position embedding approach does not enable transformers to extrapolate to inputs longer than the ones they observed during training. We then showed that extrapolation in transformers can be enabled by changing the position method for one that allows for extrapolation. We showed that our ALiBi method is an extremely simple replacement for existing position approaches that allow models to extrapolate. In addition, when not extrapolating, our method achieves either better perplexity than the sinusoidal method (in models smaller than 1B parameters, trained on less data) or similar perplexity (in larger, billion parameter models trained on much more data). Our method is simple to implement and does not slow down runtime or require extra parameters (but does sometimes require a negligible amount of extra memory). We show that using our method can speed up the training of a 1.3 billion parameter model evaluated on the same input sequence length as GPT-3 (2048).

Acknowledgments

We thank Tim Dettmers, Gabriel Ilharco, Jungo Kasai, Hao Peng, Sewon Min, Sam Shleifer, Luke Zettlemoyer, Julian Michael, Nikolaos Pappas, and Yizhong Wang for their valuable feedback and fruitful discussions.

References

Alexei Baevski and Michael Auli. Adaptive input representations for neural language modeling. *CoRR*, abs/1809.10853, 2018. URL `http://arxiv.org/abs/1809.10853`.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv:2004.05150*, 2020.

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.

Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. Unsupervised cross-lingual representation learning at scale. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2020. doi: 10.18653/v1/2020.acl-main.747. URL `http://dx.doi.org/10.18653/v1/2020.acl-main.747`.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc Le, and Ruslan Salakhutdinov. Transformer-XL: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 2978–2988, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1285. URL `https://aclanthology.org/P19-1285`.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL `https://www.aclweb.org/anthology/N19-1423`.

Aaron Gokaslan and Vanya Cohen. Openwebtext corpus. `http://Skylion007.github.io/OpenWebTextCorpus`, 2019.

Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Ian Simon, Curtis Hawthorne, Noam M. Shazeer, Andrew M. Dai, M. Hoffman, M. Dinculescu, and D. Eck. Music transformer: Generating music with long-term structure. In *ICLR*, 2019.

Hakan Inan, Khashayar Khosravi, and Richard Socher. Tying word vectors and word classifiers: A loss framework for language modeling. In *ICLR*, 2017. URL `https://openreview.net/forum?id=r1aPbsFle`.

Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. Generalization through Memorization: Nearest Neighbor Language Models. In *International Conference on Learning Representations (ICLR)*, 2020.

Mike Lewis, Shruti Bhosale, Tim Dettmers, Naman Goyal, and Luke Zettlemoyer. Base layers: Simplifying training of large, sparse models, 2021.

Opher Lieber, Or Sharir, Barak Lenz, and Yoav Shoham. Jurassic-1: Technical details and evaluation. Technical report, AI21 Labs, August 2021.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach, 2019.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models, 2016.

Tomas Mikolov and G. Zweig. Context dependent recurrent neural network language model. *2012 IEEE Spoken Language Technology Workshop (SLT)*, pp. 234–239, 2012.

Tomas Mikolov, M. Karafiát, L. Burget, J. Cernocký, and S. Khudanpur. Recurrent neural network based language model. In *INTERSPEECH*, 2010.

Sebastian Nagel. Cc-news. `https://commoncrawl.org/2016/10/news-dataset-available/`, 2016.

Myle Ott, Sergey Edunov, David Grangier, and Michael Auli. Scaling neural machine translation. In *Proceedings of the Third Conference on Machine Translation (WMT)*, 2018.

Ofir Press and Lior Wolf. Using the output embedding to improve language models. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pp. 157–163, Valencia, Spain, April 2017. Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/E17-2025`.

Ofir Press, Noah A. Smith, and Omer Levy. Improving transformer models by reordering their sublayers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 2996–3005, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.270. URL `https://www.aclweb.org/anthology/2020.acl-main.270`.

Ofir Press, Noah A. Smith, and Mike Lewis. Shortformer: Better language modeling using shorter inputs. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 5493–5505, Online, August 2021. Association for Computational Linguistics. URL `https://aclanthology.org/2021.acl-long.427`.

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

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020. URL `http://jmlr.org/papers/v21/20-074.html`.

Aurko Roy, Mohammad Saffar, Ashish Vaswani, and David Grangier. Efficient content-based sparse attention with routing transformers, 2020.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pp. 464–468, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-2074. URL `https://www.aclweb.org/anthology/N18-2074`.

Jianlin Su, Yu Lu, Shengfeng Pan, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding, 2021.

Trieu H. Trinh and Quoc V. Le. A simple method for commonsense reasoning, 2018.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL `https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf`.

Ben Wang and Aran Komatsuzaki. GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model. `https://github.com/kingoflolz/mesh-transformer-jax`, May 2021.

Ulme Wennberg and Gustav Eje Henter. The case for translation-invariant self-attention in transformer-based language models, 2021.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 38–45, Online, October 2020. Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/2020.emnlp-demos.6`.

Chuhan Wu, Fangzhao Wu, and Yongfeng Huang. DA-transformer: Distance-aware transformer. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 2059–2068, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.166. URL `https://aclanthology.org/2021.naacl-main.166`.

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization, 2014.

Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the IEEE international conference on computer vision*, pp. 19–27, 2015.

A Appendix

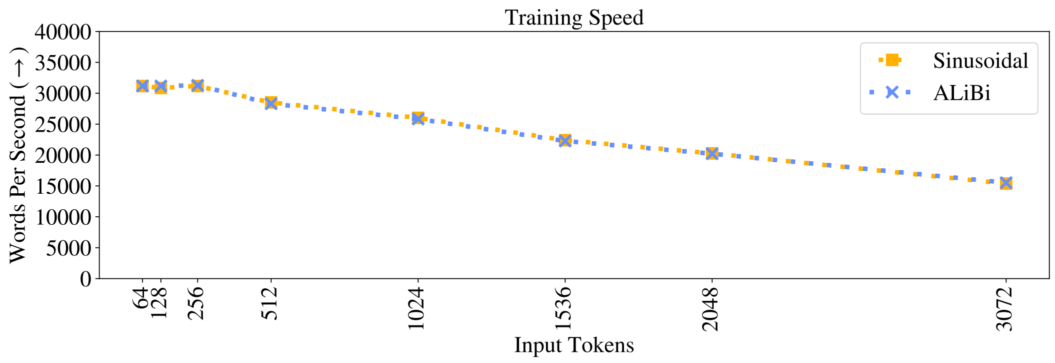

Figure 11: Training speed of our model and the sinusoidal baseline trained on different amounts of input subsequence tokens L.

Table 1: The speed (during trainining and evaluation) and memory usage (during training) of the rotary, T5 bias, and ALiBi compared to the sinusoidal baseline, on WikiText-103. Training and inference is batched and speeds are shown for one V100 GPU.

Position Method	Train Length	Speed (↑)		Memory (↓)
		Train	Eval.	
Sinusoidal	512	28.5k	82.1k	15.3
	1024	26.0k	77.8k	19.2
	3072	15.3k	42.4k	15.1
Rotary	512	20.0k	43.4k	17.8
	1024	17.7k	39.4k	22.8
	3072	11.5k	29.5k	17.8
T5 Bias	512	14.4k	21.8k	16.9
	1024	13.0k	20.2k	20.9
	3072	4.3k	4.9k	15.9
ALiBi	512	28.3k	85.8k	15.3
	1024	25.8k	76.4k	19.3
	3072	15.5k	42.2k	15.2

Table 2: The sinusoidal, rotary, T5 bias and ALiBi models, trained on L = **512**, on WikiText-103, evaluated with different values of L_{valid} on the validation set. Bold is the best score for each model. Inference speeds are from inference on a CPU with batch size of one.

	Sinusoidal		Rotary		T5 Bias		ALiBi	
Inputs	PPL (↓)	WPS (↑)	PPL (↓)	WPS (↑)	PPL (↓)	WPS (↑)	PPL (↓)	WPS (↑)
512	20.05	15046	20.07	10839	19.65	11724	19.73	14726
513	19.98	14925	20.01	10806	19.57	10491	19.62	14965
522	19.93	15116	20.02	11295	19.57	9970	19.64	15316
532	**19.91**	15358	19.98	10854	19.53	10382	19.61	15383
542	**19.91**	15076	19.94	10795	19.47	12270	19.57	15301
552	**19.91**	16394	19.93	12267	19.47	13000	19.54	16540
562	**19.91**	16646	19.87	12481	19.39	12201	19.49	16385
572	19.95	16934	19.83	12668	19.36	12851	19.46	16881
582	20.13	16961	19.88	12594	19.41	13904	19.48	17064
592	20.18	17243	19.84	13007	19.36	13706	19.43	17289
602	20.40	17502	19.81	12788	19.33	14102	19.38	17141
612	20.59	17637	19.81	12601	19.27	14573	19.38	17661
712	24.86	15614	**19.79**	12676	19.10	13818	19.14	15637
812	30.82	17151	20.17	13954	18.94	14377	18.99	17210
912	37.42	17200	20.73	13887	18.86	15345	18.88	17619
1012	43.54	16304	21.37	13759	18.79	14240	18.73	16059
1112	50.36	16424	22.01	13891	**18.77**	14014	18.68	16659
1212	58.01	17294	23.02	15245	18.87	14589	18.67	17372
1312	63.62	15314	23.93	13698	18.84	13138	18.60	15698
1412	70.75	15663	24.81	13928	18.87	12857	18.59	15860
1512	76.23	15812	25.99	14248	18.91	13752	18.52	16225
2512	132.41	15254	31.58	13456	20.41	9948	18.41	15204
3512	178.97	13293	35.54	11850	22.91	7847	18.40	13329
4512	209.37	11767	39.15	10485	25.91	6146	18.41	11738
5512	240.44	10168	43.14	9020	29.54	5309	18.36	9986
6512	271.40	9052	47.81	8108	34.48	4680	18.35	9022
7512	293.02	8315	51.12	7483	39.29	4102	18.33	8324
8512	305.65	7259	54.98	6718	43.08	3660	18.34	7366
9512	336.02	6672	57.85	6211	48.90	3370	18.34	6555
10512	341.53	6126	60.77	5575	52.95	3010	18.32	6030
11512	362.74	5994	66.62	5445	61.38	2873	18.32	5882
12512	373.17	5421	69.70	4988	64.94	2602	**18.31**	5287
13512	382.91	5174	73.27	4692	OOM	-	**18.31**	4962
14512	399.98	4351	75.52	4103	OOM	-	**18.31**	4352
15512	406.01	4291	79.25	3969	OOM	-	**18.31**	4289

Table 3: The sinusoidal, rotary, T5 bias and ALiBi models, trained on L = **1024**, on WikiText-103, evaluated with different values of L_{valid} on the validation set. Bold is the best score for each model. Inference speeds are from inference on a CPU with batch size of one.

	Sinusoidal		Rotary		T5 Bias		ALiBi	
Inputs	PPL (↓)	WPS (↑)	PPL (↓)	WPS (↑)	PPL (↓)	WPS (↑)	PPL (↓)	WPS (↑)
1024	19.34	17002	19.33	14690	18.80	14973	18.66	16951
1025	19.33	16630	19.34	14423	18.82	14635	18.67	16690
1034	19.27	16589	19.28	14351	18.74	14435	18.60	16707
1044	19.26	16760	19.27	14491	18.72	14644	18.60	16667
1054	19.23	16747	19.26	14503	18.71	14800	18.58	16833
1064	19.21	16676	19.22	14623	18.70	14498	18.55	16941
1074	**19.19**	16879	19.19	14464	18.65	14670	18.49	16936
1084	19.22	16942	19.23	14650	18.70	14607	18.56	17090
1094	19.24	16771	19.22	14629	18.69	14517	18.54	16880
1104	19.28	16870	19.27	14837	18.69	14635	18.52	17009
1114	19.29	16795	19.27	14879	18.69	14540	18.52	17050
1124	19.26	17312	**19.18**	15121	18.62	14480	18.46	17571
1224	20.54	17901	19.38	15584	18.58	14956	18.40	18013
1324	23.13	16308	19.96	14386	18.52	13726	18.33	16422
1424	26.45	16217	21.27	14385	18.48	13516	18.28	16121
1524	29.82	16377	22.59	14693	18.42	13587	18.22	16659
1624	34.27	15928	24.34	14228	18.40	12979	18.17	16053
1724	38.24	16640	25.66	14686	18.35	12976	18.15	16607
1824	42.23	16840	27.63	14918	**18.30**	13071	18.08	16846
1924	46.46	15071	29.64	13452	18.31	11843	18.08	15118
2024	51.09	15591	31.17	13706	18.34	11906	18.05	15557
3024	96.46	13639	35.67	12256	18.62	8480	**17.92**	13668
4024	144.00	12441	44.30	11203	19.44	7443	17.95	12402
5024	182.31	11431	48.31	10324	20.47	6384	**17.92**	11394
6024	214.02	10238	54.78	9117	21.76	5577	18.01	10119
7024	261.86	8785	62.83	7950	23.64	4867	17.93	8779
8024	284.88	8132	64.91	7355	25.79	4377	17.96	8086
9024	310.04	7045	71.91	6380	27.54	3787	17.98	7001
10024	337.48	6633	77.70	6016	29.54	3582	17.97	6583
11024	358.43	5722	81.15	5219	31.94	3170	18.02	5641
12024	375.95	5560	87.51	5072	33.35	2940	18.01	5294
13024	393.57	4691	94.74	4383	OOM	-	17.98	4621
14024	403.52	4905	96.10	4546	OOM	-	18.01	4827
15024	431.66	4518	99.78	4170	OOM	-	17.96	4447
16024	453.32	4239	106.99	3878	OOM	-	17.98	4153

Table 4: The sinusoidal, rotary, T5 bias and ALiBi models, trained on L = **3072**, on WikiText-103, evaluated with different values of L_{valid} on the validation set. Bold is the best score for each model. Inference speeds are from inference on a CPU with batch size of one.

	Sinusoidal		Rotary		T5 Bias		ALiBi	
Inputs	PPL (↓)	WPS (↑)	PPL (↓)	WPS (↑)	PPL (↓)	WPS (↑)	PPL (↓)	WPS (↑)
3072	18.67	13380	18.57	12548	18.01	8828	17.60	13866
3073	18.67	13773	18.57	12474	18.01	8483	17.59	13793
3082	18.62	13741	18.54	12388	17.95	8698	17.59	13778
3092	**18.60**	13742	18.48	12458	17.92	8361	17.55	13783
3102	18.65	13701	18.52	12365	17.94	8764	17.59	13747
3112	18.64	13809	18.51	12449	17.96	8665	17.59	13827
3122	18.68	13722	18.52	12432	17.98	8437	17.58	13795
3132	18.67	13825	18.54	12490	17.97	8653	17.58	13784
3142	18.69	13543	18.52	12230	17.97	8282	17.61	13572
3152	18.66	13520	18.56	12240	17.98	8608	17.59	13523
3162	18.71	13501	18.56	12253	18.04	8589	17.62	13598
3172	18.72	13563	18.55	12297	17.99	8583	17.59	13625
3272	18.87	13453	18.55	12148	17.93	8144	17.59	13482
3372	19.46	13533	18.50	12254	17.88	8442	17.52	13565
3472	20.55	13047	18.52	11868	17.95	7857	17.54	13107
3572	21.84	13128	18.50	11882	17.86	7814	17.50	13170
3672	23.04	13106	18.49	11859	17.87	7719	17.48	13196
3772	24.47	13287	18.54	11942	17.85	7579	17.49	13312
3872	25.85	12621	**18.40**	11272	17.82	7581	17.41	12566
3972	27.21	12379	18.48	11151	17.84	7483	17.41	12324
4072	28.59	12178	18.59	11019	17.88	6974	17.48	12212
5072	45.53	11076	18.80	9887	17.76	6230	17.33	10938
6072	65.01	10114	19.50	9049	**17.68**	5554	17.26	10133
7072	85.96	8647	20.60	7861	17.83	4820	17.22	8670
8072	102.74	7755	21.60	6991	18.06	4281	17.30	7729
9072	125.99	6953	22.14	6360	18.12	3823	17.26	6939
10072	133.68	6646	23.21	6068	18.37	3579	17.28	6597
11072	161.29	5663	24.39	5158	18.64	3119	17.26	5585
12072	169.55	5567	26.70	5111	18.93	2920	17.24	5397
13072	189.43	5044	29.33	4658	19.10	2735	**17.15**	4809
14072	203.86	4915	32.21	4616	OOM	-	17.22	4866
15072	221.14	4561	33.47	4292	OOM	-	17.23	4491
16072	231.29	4382	34.51	4099	OOM	-	17.22	4312

Table 5: ALiBi extrapolating on the WikiText-103 development set. *For the results we present for the sinusoidal, rotary and T5 bias models, $L = L_{valid}$ (so we do not test the extrapolation abilities of those baselines here).

ALiBi	Evaluation Length							
Train Length	64	128	256	512	1024	1536	2048	3072
64	28.46	24.70	22.88	22.09	21.73	21.63	21.59	21.53
128	-	23.98	21.70	20.67	20.36	20.29	20.31	20.28
256	-	-	**21.29**	19.89	19.29	19.13	19.10	19.03
512	-	-	-	19.73	18.81	18.50	18.48	18.40
1024	-	-	-	-	**18.66**	18.20	18.05	17.96
1536	-	-	-	-	-	**18.12**	17.90	17.72
2048	-	-	-	-	-	-	**17.91**	17.64
3072	-	-	-	-	-	-	-	**17.60**
Sinusoidal*	**28.03**	**23.81**	21.45	20.05	19.34	19.05	18.87	18.67
Rotary*	-	-	-	20.07	19.33	-	-	18.57
T5 Bias*	-	-	-	**19.65**	18.80	-	-	18.01

Table 6: Test perplexity and runtime on WikiText-103 for two of our ALiBi models and models that use the sinusoidal, rotary and T5 bias methods.

	Model	Param. ↓	Train	Inference		
			Speed↑	Speed ↑	Valid ↓	Test ↓
	Sinusoidal, L = 3072	**247M**	15.3k	**13.6k**	18.67	19.38
	Rotary, L = 3072	**247M**	**11.5k**	12.2k	18.57	19.28
	T5 Bias, L = 3072	**247M**	4.3k	7.3k	18.01	18.73
ALiBi	L = 512, L_{valid} = 3072	**247M**	28.3k	**13.6k**	18.40	19.08
ALiBi	L = 3072, L_{valid} = 3072	**247M**	15.5k	**13.6k**	**17.60**	**18.30**

Table 7: Valid and test perplexity scores on WikiText-103 for two of our ALiBi models and models that use the sinusoidal, rotary and T5 bias methods, with sliding window evaluation (§5, with S=512 following (Baevski & Auli, 2018; Khandelwal et al., 2020; Press et al., 2021)). The "sinusoidal" model presents our results from training and inference with the model of Baevski & Auli.

	Model	Param. ↓	Valid ↓	Test ↓
	Adaptive Inputs (Baevski & Auli, 2018)	**247M**	17.97	18.70
	Transformer-XL (Dai et al., 2019)	257M	-	18.3
	Shortformer (Press et al., 2021)	**247M**	17.47	18.15
	Sandwich Transformer (Press et al., 2020)	**247M**	-	17.96
	Staged Training (Press et al., 2021)	**247M**	-	17.56
	Compressive Transformer (Rae et al., 2020)	329M	-	17.1
	Routing Transformer (Roy et al., 2020)	-	-	15.8
	kNN-LM (Khandelwal et al., 2020)	**247M**	15.81	**15.79**
	Sinusoidal, L = 3072	**247M**	17.95	18.67
	Rotary, L = 3072	**247M**	17.98	18.72
	T5 Bias, L = 3072	**247M**	17.37	18.12
ALiBi	L = 512, L_{valid} = 3072	**247M**	18.30	19.01
ALiBi	L = 3072, L_{valid} = 3072	**247M**	16.97	17.66

Table 8: ALiBi models extrapolating on the Toronto BookCorpus development set. *For the results of the sinusoidal models, L = L_{valid} (so we do not test the extrapolation abilities of those models here).

	Evaluation Length		
Train Length	512	1024	3072
512	14.29	13.64	13.55
1024	-	13.86	13.52
3072	-	-	13.15
Sinusoidal*	14.80	14.73	14.46

Table 9: Valid and test perplexities on the Toronto Book Corpus dataset.

	Model	Param. ↓	Valid ↓	Test ↓
	Sinusoidal, L = 3072	**247M**	14.46	11.67
ALiBi	L_{train} = 512, L_{valid} = 3072	**247M**	13.55	10.98
	L_{train} = 3072, L_{valid} = 3072	**247M**	**13.15**	**10.73**

Table 10: Valid and test perplexities on the Toronto Book Corpus dataset, with a sliding window (§5). Following (Baevski & Auli, 2018; Khandelwal et al., 2020; Press et al., 2020; 2021) we set the sliding window stride S=512.

	Model	Param. ↓	Valid ↓	Test ↓
	kNN-LM (Khandelwal et al., 2020)	**247M**	14.20	10.89
	Shortformer (Press et al., 2021)	**247M**	13.40	10.88
	Sandwich (Press et al., 2020)	**247M**	-	10.83
	Staged Training (Press et al., 2021)	**247M**	12.80	10.48
	Sinusoidal, L = 3072	**247M**	14.06	11.40
ALiBi	L = 512, L_{valid} = 3072	**247M**	13.76	11.11
	L = 3072, L_{valid} = 3072	**247M**	12.70	10.40

Table 11: Perplexity, memory, and train time on the CC100+RoBERTa corpus, for our ALiBi models and the sinusoidal baseline. We run our L = 512 (1024) ,pde; and the sinusoidal model with L = 1024 (2048) for the same amount of time, and show that our models achieve strong results even though they use 6%-11% less memory.

	Training			Valid PPL ↓	
	Memory ↓	Updates	Hours ↓	L_{valid} = 1024	L_{valid} = 2048
Sinusoidal, L_{train} = 1024	26.2 GB	46.7k	5.5k	**9.24**	-
ALiBi, L_{train} = 512	**24.6 GB**	50.0k	5.5k	9.30	-
Sinusoidal, L_{train} = 2048	29.3 GB	44.2k	5.9k	-	9.01
ALiBi, L_{train} = 1024	**26.2 GB**	50.0k	5.9k	-	**8.92**

Table 12: Perplexity, train time and memory use of the sinusoidal and ALiBi models on the CC100+RoBERTa corpus, when all models are trained with 50k updates.

	Training			Valid PPL ↓		
	Memory ↓	Updates	Hours ↓	L_{valid} = 512	L_{valid} = 1024	L_{valid} = 2048
Sinusoidal, L_{train} = 512	24.6GB	50.0k	5.5k	9.71	-	-
ALiBi, L_{train} = 512	24.6GB	50.0k	5.5k	9.79	-	-
Sinusoidal, L_{train} = 1024	26.2GB	50.0k	5.9k	-	9.15	-
ALiBi, L_{train} = 1024	26.2GB	50.0k	5.9k	-	9.16	-
Sinusoidal, L_{train} = 2048	29.3GB	50.0k	6.7k	-	-	8.83
ALiBi, L_{train} = 2048	29.4GB	50.0k	6.7k	-	-	8.84

Table 13: **Sinusoidal** models evaluated with sliding window evaluation with stride $S = 1$ on the WikiText-103 validation dataset.

	Evaluation Length ($S = 1$)				
Train Length	512	1024	1536	2048	3072
512	18.35	204.42	264.74	306.19	360.12
1024	-	18.05	206.55	302.6	393.71
3072	-	-	-	-	18.03

Table 14: **ALiBi** models evaluated with sliding window evaluation with stride $S = 1$ on the WikiText-103 validation dataset.

	Evaluation Length ($S = 1$)				
Train Length	512	1024	1536	2048	3072
512	17.98	17.92	18.2	18.28	18.3
1024	-	17.46	17.47	17.62	17.92
3072	-	-	-	-	16.96

Table 15: **T5 bias** models evaluated with sliding window evaluation with stride $S = 1$ on the WikiText-103 validation dataset.

	Evaluation Length ($S = 1$)				
Train Length	512	1024	1536	2048	3072
512	17.92	18.51	20.36	22.62	30.77
1024	-	17.65	17.87	18.51	20.66
3072	-	-	-	-	17.41