# ROFORMER:

## Enhanced Transformer with Rotary Position embedding

Presented by

**Maryam Bacchus**

Jan 20, 2026

## Problem and Relevance

- In transformers, the self attention mechanism doesn’t encode word positions.
- Word position in sentences matter.

- Without encoding position data
  - “The cat eats a rat” == “A rat eats the cat”
  - The two sentences contain the same words, however the position conveys different meaning. In standard self attention, the two sentences are equal in meaning.

1. Absolute position embedding

- Used in BERT, GPT-2 and early transformer models
- Each position has an assigned fixed vector eg. position1 = [0,0,0]
- Position encodings are done by adding the word vector to the position vector
  - $$f_{t:t\in\{q,k,v\}}(\boldsymbol{x}_i, i) := \boldsymbol{W}_{t:t\in\{q,k,v\}}(\boldsymbol{x}_i + \boldsymbol{p}_i)$$

Problem

it does not encode information about how far away one word is from another. It just knows the position of a word.

---

2. Relative position embedding

- Encodes information about how far apart each word in a sentence is from each other.
- Example: “the cat eats”
  - the → cat = 1 position
  - the → eats = 2 positions
  - cat → the = -1 positions

Problem
each word has a matrix representation of its relationship to every single other word.
O(n^2) complexity

**Proposes Approach - RoPE - Rotary Position Embedding**

---

- Solves the issue by adding rotational values to each vector embedding
- Multiplies each word by a rotation matrix
- Example “The cat drinks”
  - The → rotate by 1 degree
  - Cat → rotate by 2 degree
  - Drinks → rotate by 3 degree
- When computing attention, the math automatically gives you relative position (by subtracting two vectors, you get the distance between them)

$$f_{\{q,k\}}(\boldsymbol{x}_m, m) = \begin{pmatrix} \cos m\theta & -\sin m\theta \\ \sin m\theta & \cos m\theta \end{pmatrix} \begin{pmatrix} W^{(11)}_{\{q,k\}} & W^{(12)}_{\{q,k\}} \\ W^{(21)}_{\{q,k\}} & W^{(22)}_{\{q,k\}} \end{pmatrix} \begin{pmatrix} x^{(1)}_m \\ x^{(2)}_m \end{pmatrix}$$

# Proposes Approach - RoPE - Rotary Position Embedding

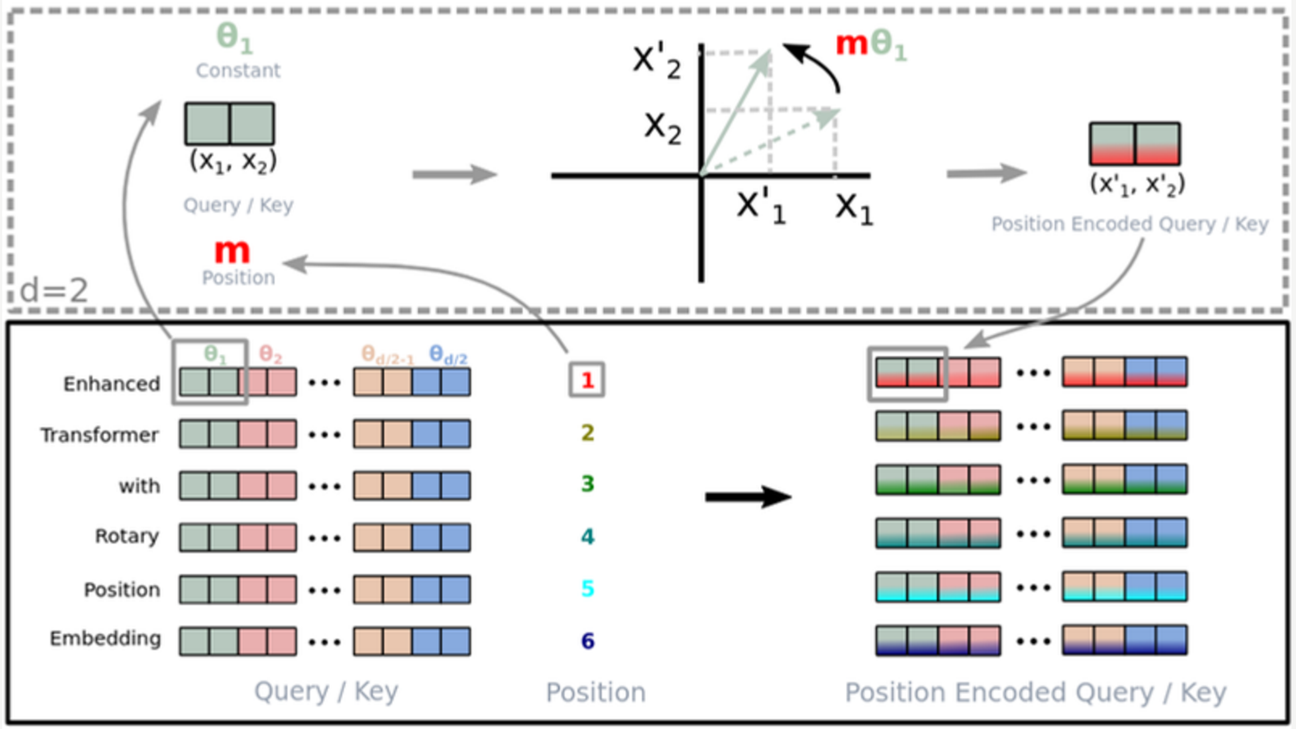



Figure 1: Implementation of Rotary Position Embedding(RoPE).

***Visual representation of rotatatory position embedding***

## Properties of RoPE

- Long Term Decay
  - The further apart two words are, the less they pay attention to each other. Vectors might point in different directions when at a significantly large distance away.

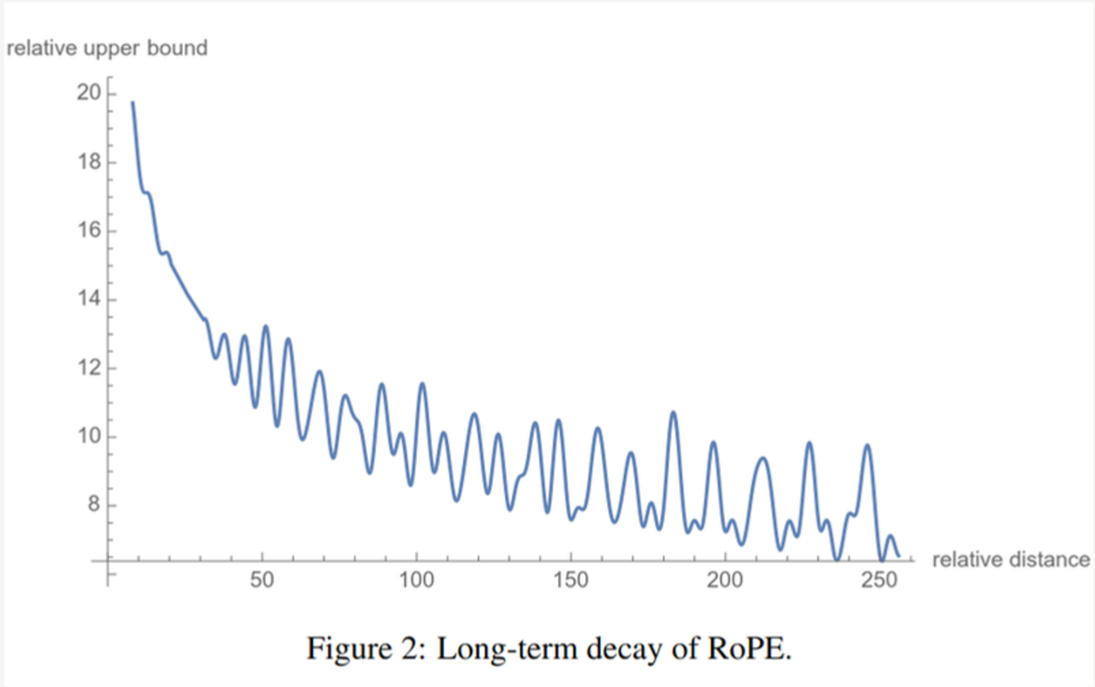



Figure 2: Long-term decay of RoPE.

## Properties of RoPE

- Linear Attention
  - RoPE applies rotation before attention, directly to Q and K in the vector embedding.
  - Solves the problem of losing context when using different attention mechanisms

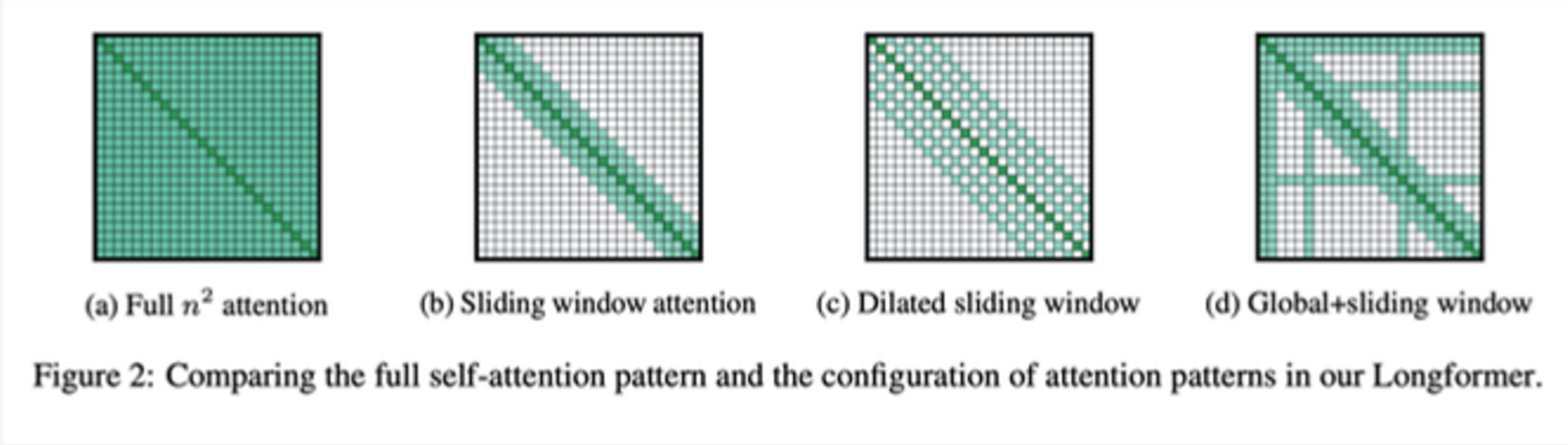



Figure 2: Comparing the full self-attention pattern and the configuration of attention patterns in our Longformer.

## Computational Efficeincy of RoFormer - Performer with RoPE

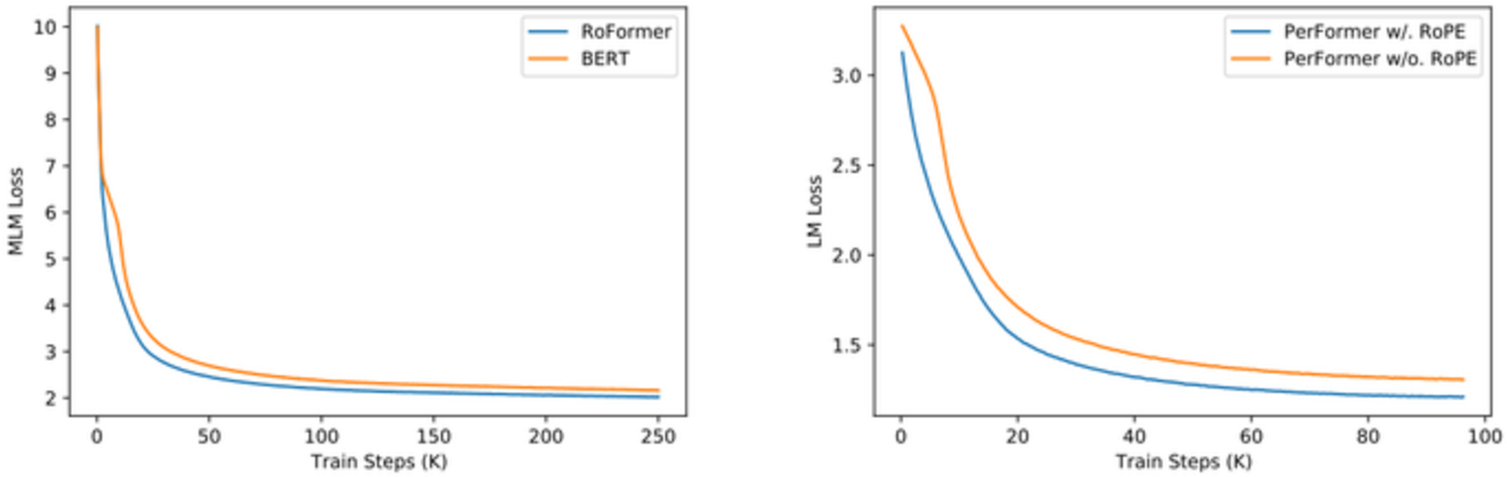



Figure 3: Evaluation of RoPE in language modeling pre-training. **Left**: training loss for BERT and RoFormer. **Right**: training loss for PerFormer with and without RoPE.

*The MLM loss during pre-training is shown on the left plot of Figure (3). Compare to the vanilla BERT, RoFormer experiences faster convergence.*

# *ALiBi*

## Train Short, Test Long: Attention with Linear Biases Enables Input Length Extrapolation (ALiBi)

Presented by

**Maryam Bacchus**

**Jan 20, 2026**

## Definitions

- Context length - the maximum number of tokens the model attention can process at once.
- Sequence length - the number of tokens of an input/output.
- Sequence length > Context length = the model is confused (high perplexity score)

## Problem and Relevance

- Transformers are terrible at extrapolation.
- At inference, if the model sees a longer sequence length, than it expects, it gets confused.
- Example
  - In training, you use a context size of 512 . If you pass 1000 tokens at inference, then the model doesn’t know how to interpret the information.
  - The model only learned patterns, relationships and how to use attention within the 512 token length constraint.

- Problem
  - You can increase the context length by increasing the sequence length during training to improve predictions, but longer sequences = more expenses (memory and time).

# Problem

The training speed of transformer LMs gets slower as the input subsequence length $L$ increases. Figure 7 visualizes this.

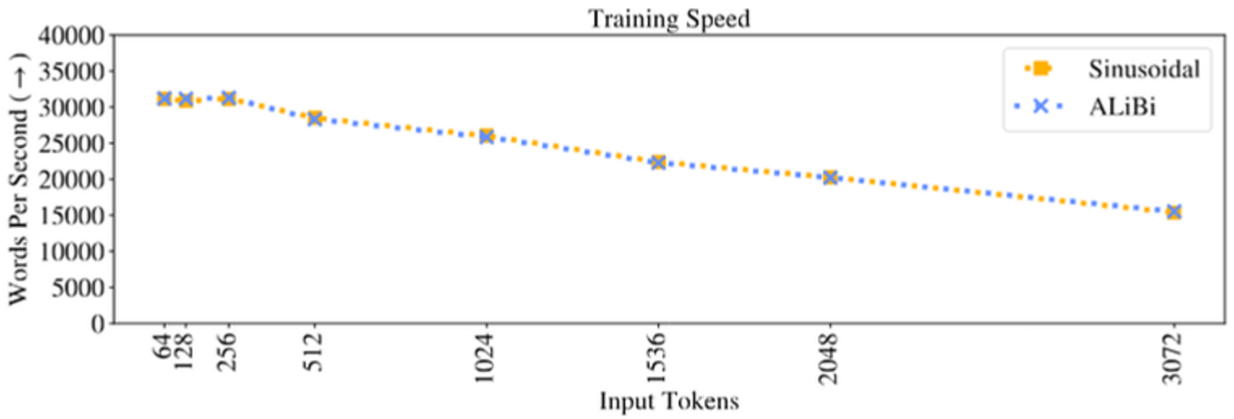



Figure 7: Training speed of our model and the sinusoidal baseline trained on different amounts of input subsequence tokens $L$.

## Background and Related Works

1. Extrapolation is poor because of position embedding

- Rotary position embedding and sinusoidal position embedding limits extrapolation.
- T5 bias (which is a relative position embedding type) performs better at extrapolation than the former two, however it is computationally expensive.

## Proposed Approach - ALibi - Attention with Linear Biases

Explanation

- In the transformer model, position embedding are added to the word embeddings at the bottom of the network.
- For an input sub-sequence of length L, each token only attends to a subset of the sequence - itself and everything before it. $\mathbf{q}_i \in \mathbb{R}^{1\times d}, (1 \leq i \leq L)$
- Softmax calculation
- The attention scores are then multiplied together and the output is returned.
- Because of the way attention is calculated, the model can only learn to understand sequences =< than the context length

## Proposed Approach - ALibi - Attention with Linear Biases

---

ALibi

- Do not add position embeddings at any point in the network.
- ALiBi adds a static, non-learned bias to the attention scores before the softmax is applied.
- Has natural tendency to penalizes attention scores between distant query-key pairs
- Penalty increases as distance between key and query increases

## Proposed Approach - ALibi - Attention with Linear Biases

# Explanation

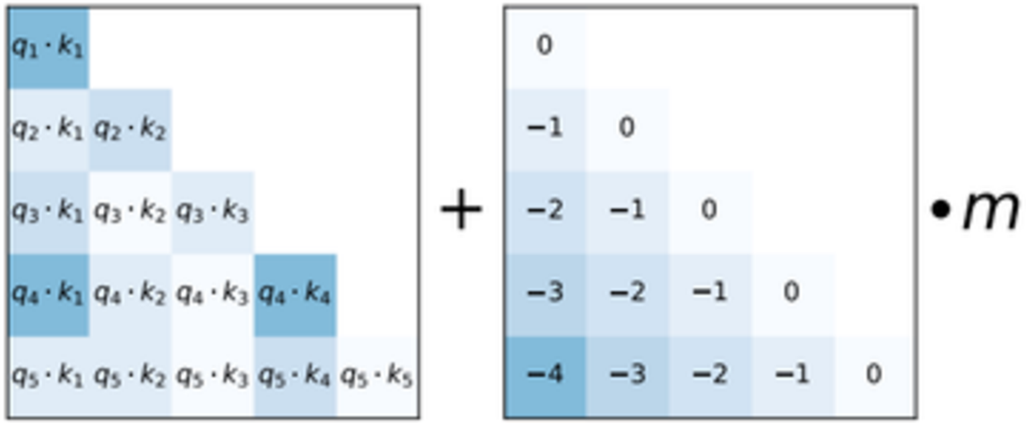



Figure 3: When computing attention scores for each head, our linearly biased attention method, ALiBi, adds a constant bias (right) to each attention score ($\mathbf{q}_i \cdot \mathbf{k}_j$, left). As in the unmodified attention sublayer, the softmax function is then applied to these scores, and the rest of the computation is unmodified. **m is a head-specific scalar** that is set and not learned throughout training. We show that our method for setting $m$ 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.

## Experiment Setup

- Using a transformer model
- Training set: WikiText-103 corpus (103 million tokens)
- Variables
  - Position embedding methods
  - Sequence length

## Results

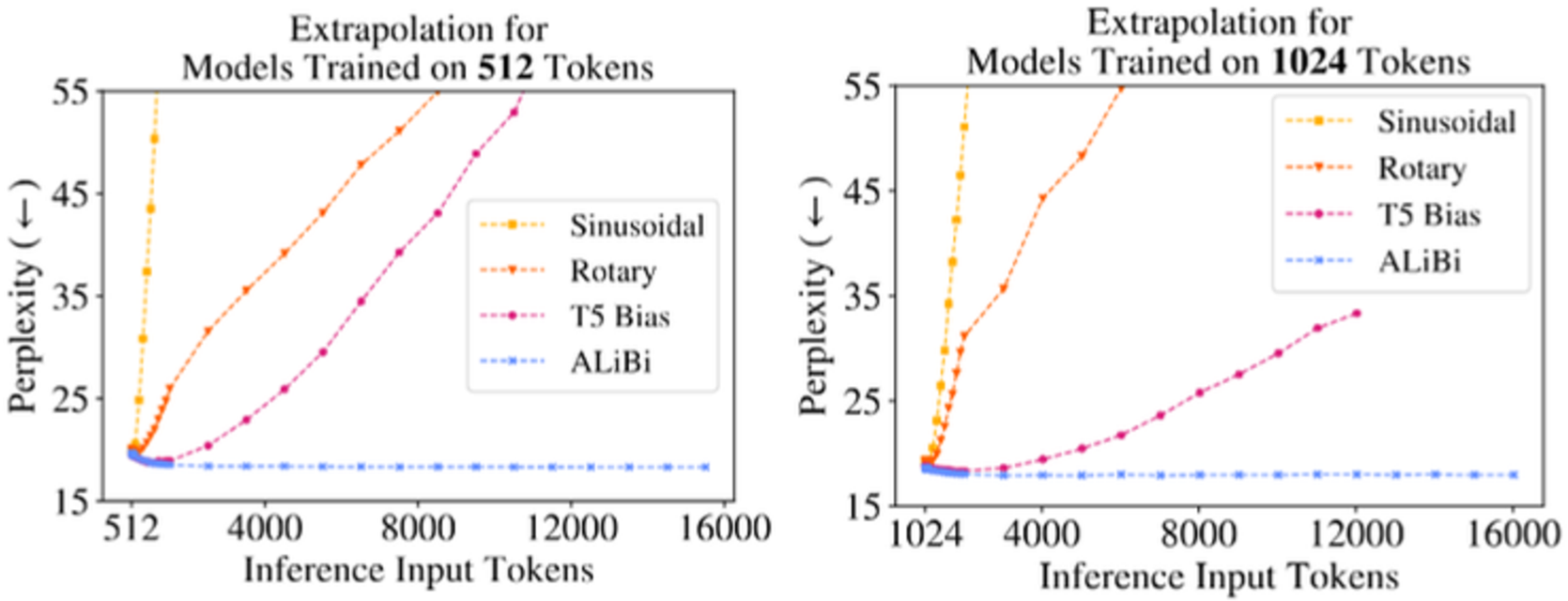
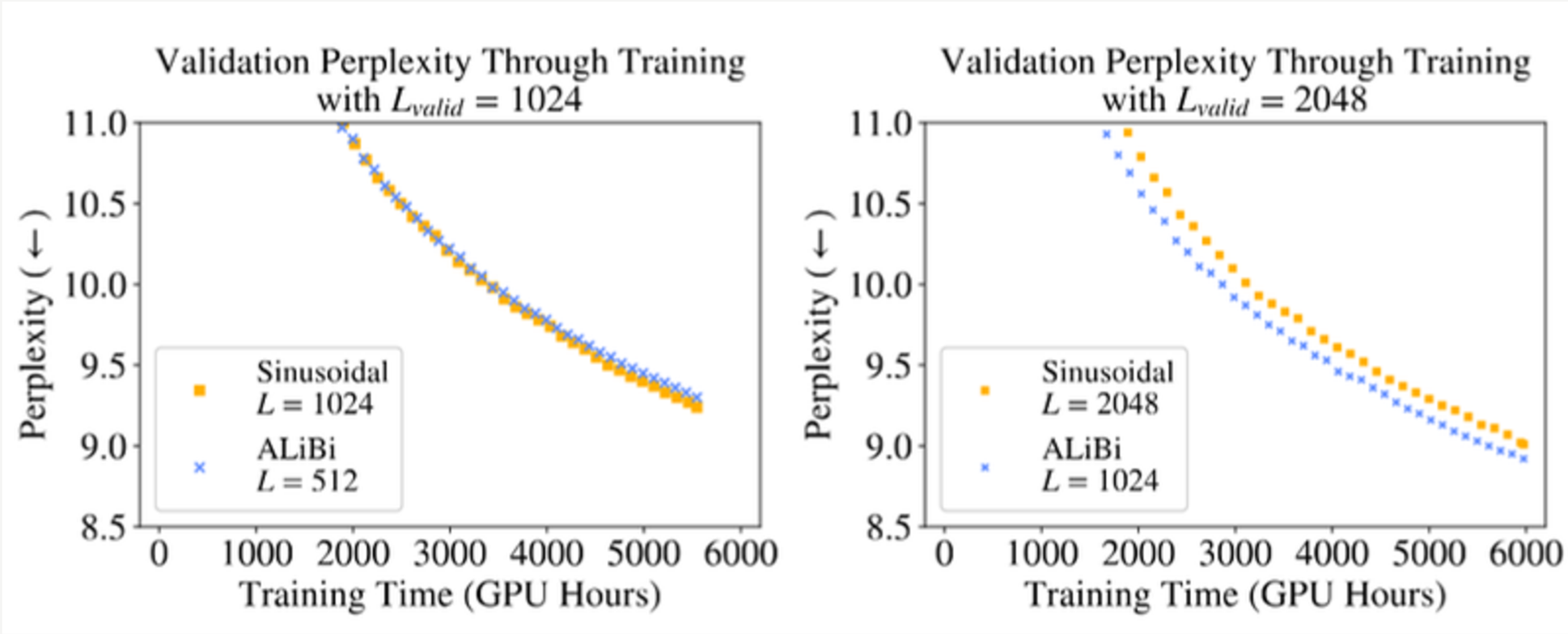



Figure 1: Extrapolation: as the (validation-set's) input sequence gets longer ($x$-axis), current position methods (sinusoidal, rotary, and T5) show degraded perplexity ($y$-axis, lower is better), but our method (§3) does not. Models were trained on WikiText-103 with sequences of $L = 512$ (left) or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

## Results


Validation Perplexity Through Training with $L_{valid} = 1024$
Perplexity (↓)
11.0
10.5
10.0
9.5
9.0
8.5
0
1000
2000
3000
4000
5000
6000
Training Time (GPU Hours)
Sinusoidal $L = 1024$
ALiBi $L = 512$
Validation Perplexity Through Training with $L_{valid} = 2048$
Perplexity (↓)
11.0
10.5
10.0
9.5
9.0
8.5
0
1000
2000
3000
4000
5000
6000
Training Time (GPU Hours)
Sinusoidal $L = 2048$
ALiBi $L = 1024$

---

Results of ALibi

- Using ALiBi, a transformer LM can be trained on short-L sequences and therefore at much lower cost, and it can still be reliably applied to long sequences at runtime.
- example,
  - 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 sequences of 2048 tokens, alib is 11% faster and uses 11% less memory.